Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
Programa de Doctorat en Computació

PhD Dissertation

Geometric constraint solving in a dynamic geometry framework

Marta R. Hidalgo Garcia

Advisor
Robert Joan Arinyo

Barcelona, 2013

A Fina, que haguera disfrutat només mirant aquesta tesi
In Memoriam

Acknowledgements

La reconnaissance est la memoire du coeur.

Jean Baptiste Massieu

En primer lloc vull agraïr de tot cor a Robert Joan tota l'ajuda i dedicació que sempre m'ha prestat. Ha sabut guiar-me fins al final d'aquest llarg camí. Vull agraïr també a la resta del grup de recerca per les xerrades junts i els consells que sempre han estat disposats a donar-me.

Gràcies a Dominique Michelucci per acollir-me a Dijon i donar-me l'oportunitat de treballar al seu laboratori. Gràcies també a tots els companys del laboratori Le2i i als companys de despatx Jean-Marc, Arnaud, Tomas, Abdoulaye, i en especial a George, Vishal, Cyril i Laureline per fer la meva estada molt més agradable.

M'agradaria donar les gràcies a Enrique Zuazua i a Francisco Palacios per confiar en mi i deixar-me formar part del seu projecte. Segurament, aquesta tesi va començar gràcies a ells. També a tots els companys de l'IMDEA Matemáticas, en especial a Fátima, José María, Markus i Sebastian.

Vull també donar les gràcies a tots els professors que he tingut des de petita i que m'han ensenyat a estimar les seves assignatures. En especial a Vicent Teruel per descobrir-me les matemàtiques i a Joan L. Monterde per ensenyar-me la bellesa de la geometria. Vull agraïr especialment a Manolo Sanchis tota la seva ajuda i suport a l'hora de fer la meva tesis de Màster.

Gràcies a tota la secció del Departament de Llenguatges i Sistemes Informàtics per acollir-me i fer-me sentir una més del grup, i als companys de despatx que he tingut durant aquests anys: Eduard, Eloi, Irving, Jonàs, Marc, Pasku, Sergio i en especial a Sergi, per ser un amic a més d'un company tot aquest temps.

Gràcies a tots els meus amics. A Mireia i Helena per voler estar sempre en el mateix metre quadrat que jo. A Eva, per estar cada dilluns a l'altre costat de l'Skype. A Natalia i Pili i la resta de companys del cineclub. A Àngels, Marina, Rafel, les dues Maries, Alfonso, Marta i en especial a Carlos, per l'any inoblidable que vam passar junts a Mainz. A Joan i Gabo i la resta de companys de Matemàtiques. A Anna i Jaume per estar ahí des de petits. I a tota la resta de gent que m'ha recolzat i m'ha deixat formar part de la seva vida.

Vull donar les gràcies a tota la meva família, dispersada per tota la geografia espanyola des de Vigo fins a Motril, passant per El Bonillo, Caudete, La Vila Joiosa i els voltants de València. En especial a Rosario, que em va obrir les portes de sa casa, i a les meves àvies Rosa i Fina, dones treballadores en temps difícils, per l'exemple que m'han donat.

Vull agraïr especialment als meus pares haver-me recolzat en totes les decisions que he pres en la meva vida, per difícils que els resultaren. Mai els podré estar suficientment agraïda.

Per últim vull donar les gràcies a Toni per confiar sempre en mi, perquè sé que sense ell aquesta tesi no haguera existit. I per donar alegria i pau a la meva vida cada dia.

Abstract

Geometric constraint solving is a central topic in many fields such as parametric solid modeling, computer-aided design or chemical molecular docking. A geometric constraint problem consists of a set geometric objects on which a set of constraints is defined. Solving the geometric constraint problem means finding a placement for the geometric elements with respect to each other such that the set of constraints holds.

Clearly, the primary goal of geometric constraint solving is to define rigid shapes. However an interesting problem arises when we ask whether allowing parameter constraint values to change with time makes sense. The answer is in the positive. Assuming a continuous change in the variant parameters, the result of the geometric constraint solving with variant parameters would result in the generation of families of different shapes built on top of the same geometric elements but governed by a fixed set of constraints. Considering the problem where several parameters change simultaneously would be a great accomplishment. However the potential combinatorial complexity make us to consider problems with just one variant parameter. Elaborating on work from other authors, we develop a new algorithm based on a new tool we have called h-graphs that properly solves the geometric constraint solving problem with one variant parameter. We offer a complete proof for the soundness of the approach which was missing in the original work.

Dynamic geometry is a computer-based technology developed to teach geometry at secondary schools. This technology provides the users with tools to define geometric constructions along with interaction tools such as drag-and-drop. The goal of the system is to show in the user's screen how the geometry changes in real time as the user interacts with the system. It is argued that this kind of interaction fosters students interest in experimenting and checking their ideas. The most important drawback of dynamic geometry is

that it is the user who must know how the geometric problem is actually solved. Based on the fact that current user-computer interaction technology basically allows the user to drag just one geometric element at a time, we have developed a new dynamic geometry approach based on two ideas: 1) the underlying problem is just a geometric constraint problem with one variant parameter, which can be different for each drag-and-drop operation, and, 2) the burden of solving the geometric problem is left to the geometric constraint solver.

Two classic and interesting problems in many computational models are the reachability and the tracing problems. Reachability consists in deciding whether a certain state of the system can be reached from a given initial state following a set of allowed transformations. This problem is paramount in many fields such as robotics, path finding, path planing, Petri Nets, etc. When translated to dynamic geometry two specific problems arise: 1) when intersecting geometric elements were at least one of them has degree two or higher, the solution is not unique and, 2) for given values assigned to constraint parameters, it may well be the case that the geometric problem is not realizable. For example computing the intersection of two parallel lines. Within our geometric constraint-based dynamic geometry system we have developed an specific approach that solves both the reachability and the tracing problems by properly applying tools from dynamic systems theory.

Finally we consider Henneberg graphs, Laman graphs and tree-decomposable graphs which are fundamental tools in geometric constraint solving and its applications. We study which relationships can be established between them and show the conditions under which Henneberg constructions preserve graph tree-decomposability. Then we develop an algorithm to automatically generate tree-decomposable Laman graphs of a given order using Henneberg construction steps.

Contents

List of Figures

CHAPTER 1

Introduction

The *reachability* problem is a fundamental issue in the context of many models and abstractions which describe different computational processes. Analysis of the computational traces and predictability questions for such models can be formalized as a set of different reachability problems. Reachability can be formulated in general as

> Given a computational system with a set of allowed transformations, also called functions, decide whether a certain state of a system is reachable from a given initial state by a set of allowed transformations.

A huge amount of literature on reachability have been published mainly in the field of abstract computational models see for example [10]. Examples of classical fields where the reachability problem is considered are graphs theory, [14, 93, 95, 111], Petri nets, [80, 81, 85, 110], motion planning,[54, 76, 77], geographical navigation,[2, 3, 32, 92] and robotics, [69, 77, 109, 112]. A feature common to most of the referred works deal with systems the behavior of which can be basically captured by identifying a set of well-defined discrete states.

An emerging field where the reachability problem plays an important role is dynamic geometry, [70, 108]. Dynamic geometry is a discipline that appeared during the 80's as a new tool in geometry. A number of software systems were designed for teaching geometry in secondary schools where the ruler and compass were replaced by computers featuring

high resolution color screens for user-computer interaction. The key concept in dynamic geometry is *interaction*, that is, select a geometric object in the screen, move it and see immediately how the geometric construction changes.

Compared to ruler-and-compass drawings on the paper, dynamic geometry systems offer two clear advantages. First they provide tools to create accurate drawings (intersection points, tangencies, etc). Second the computer can record the way the user constructed the geometric elements that allows it to quickly rebuild the construction every time the user changes the values assigned to some parameters. As a consequence, exploration and experimentation are encouraged by showing to the user which parts of the construction change and which remain unchanged.

In this context, a reachability problem naturally arises and can informally be stated as follows.

> Let I_s and I_e be two instances of a well defined geometric construction where I_s is called the starting instance and I_e the ending instance. Are there continuous transformations that, preserving the incidence relationships established in the geometric construction, brings I_s to I_e?

Now the scenario is different from the one described above. Notice that in dynamic geometry no set of well defined states can be identified. Moreover, the reachability problem in dynamic geometry belongs to a continuous domain.

A problem in dynamic geometry tightly related to the reachability problem is the *tracing* problem which can be informally defined as

> Let I_s and I_e be two specific instances of a well defined geometric construction where I_s is called the starting instance and I_e the ending instance. Let $I_0, I_1, \ldots, I_i, \ldots, I_n$ be a sequence of well defined geometric constructions such that I_{i+1} is a continuous incremental variation of I_i which preserves incidence relationships. If we set $I_0 = I_s$, does the sequence of incremental constructions end at $I_n = I_e$?

When using current dynamic geometry systems, the user can find that strange things happen from time to time. An object might suddenly jump into a different position or disappear completely or a group of objects may converge to the same position. It turns out that these effects are caused by a number of non-trivial mathematical issues. Since they have a dramatic effect on both reachability and tracing in dynamic geometry, they must be properly addressed.

The main sources of these problems are ambiguities and unfeasibility of geometric constructions when some parameters are changed continuously. One source of ambiguity is

the fact that, in general, geometric operations have more than one solution, for example, intersecting a line and a circle. Another ambiguity appears when a problem with a well-defined solution whenever geometric elements are in general position, say computing the point where two straight lines intersect, reaches a degenerate configuration, for example, the straight lines became parallel.

Examples of unfeasibility of the geometric construction appears whenever solving the equations underlying the geometric problem requires dividing by zero or computing square roots of negative values. The set of parameter values where constructions are unfeasible are called *critical points*.

In order to work out a satisfactory solution for both the reachability and tracing problems in dynamic geometry a well-defined method for handling both ambiguities and geometric unfeasibility must be found.

There is a paucity of works concerning reachability and tracing problems in dynamic geometry. Richter-Gebert and Kortenkamp in [90] formalized the reachability problem in computational geometry and proved that its complexity is NP-hard in $\mathbb{R}$. To deal with the tracing problem, authors describe a method based on applying a detour to the tracing whenever it gets close to conditions that, according to numerical heuristics, are close to critical points.

In [15, 16], Denner-Broser describes a decision algorithm to solve the reachability problem in dynamic geometry using Voronoi diagrams. In a first step the algorithm computes the Voronoi diagram defined by the sites corresponding to critical points. The reachability problem is solved by checking whether there is a path of Voronoi edges connecting the starting and ending points in the Voronoi partition associated to the starting and ending geometric instances such that avoids the Voronoi sites. However, no evidences of any implementation are given.

1.1 Goals

The main goal of this thesis is to establish a theoretical framework to solve the reachability and tracing problems in dynamic geometry.

As a proof of concept we aim at actually developing a software system based on our theoretical conceptualization. The system shall be built on top of a graph-based, constructive geometric constraint solving system already developed by our research group.

1.2 Scientific contributions

The scientific contributions of this work belong to one of two categories: Basic tools and main goals. Among the basic tools we find

h-graphs : We introduce a new representation for tree-decomposable Laman graphs, which we call *h-graphs*, which includes the information about its tree-decomposition and presents some nice properties. H-graphs are used with different purposes in this work.

van der Meiden soundness : We describe in detail the method to compute the domain of a geometric constraint problem with one degree of freedom reported by van der Meiden in [105]. We formalize the underlying concepts and prove for the first time that the method is correct.

Henneberg graphs : We establish some relationships between Henneberg graphs, tree-decomposable graphs and Laman graphs. We then develop a correct algorithm which computes tree-decomposable Laman graphs of a given size using Henneberg constructions. Here h-graphs play a central role.

Contributions to the main goals are

Reachability : We define a theoretical framework to solve the reachability problem in dynamic geometry. We show that the approach is correct and that it finds a solution whenever one exists. We develop a specific implementation in the framework of a dynamic geometry system based on constructive geometric constraint solving.

Tracing : We develop a solution to the tracing problem as a derivation of the solution to the reachability problem. The approach is implemented as a unit in our dynamic geometry system.

1.3 Organization of the work

This thesis includes eight chapters organized in four main parts. First, in Chapter 2 we introduce the basic concepts used in subsequent chapters. We recall elementary definitions on graphs, geometric constraint problems and dynamic geometry.

The second part includes Chapter 3 and is devoted to introduce h-graphs, a new way to represent tree-decomposable Laman graphs. h-graphs capture both a geometric constraint

problem and the associated tree-decomposition. Later on in this work, h-graphs will play a central role.

The third part includes Chapters 4, 5 and 6. It is devoted to solve the reachability and tracing problems in geometric constraint-based dynamic geometry. In Chapter 4 we prove for the first time that the van der Meiden approach to compute critical points is correct. Then we describe our own implementation based on h-graphs. In Chapter 5 we develop our approach to solve the reachability problem in dynamic geometry. A proof of the optimality of the searching algorithm is presented. We describe the prototype implemented on top of our geometric constraint-based dynamic geometry system. Chapter 6 describes our solution to the tracing problem. Some remarks about continuity in our system are highlighted. The implementation on top of our geometric constraint-based dynamic geometry system is also described.

The last part of this work includes Chapter 7. Here we develop a correct method to automatically build tree-decomposable Laman graphs of a given size using Henneberg constructions. The approach heavily relays on h-graphs.

Finally, in Chapter 8 we offer some conclusions and describe open problems we aim at exploring in the near future.

CHAPTER 2

Preliminaries

Now, in order to answer the question,
"Where do we go from here"
which is our theme, we must first
honestly recognize where we are now.

Martin Luther King

In this chapter we review some basic facts about graphs, geometric constraint solving and dynamic geometry which we will use in this work. Readers already familiarized with these fields may skip it, although we shall refer to the concepts presented here all along the manuscript.

This chapter is by no means intended to be comprehensive. Most of the concepts introduced here are explained in more detail and in a wider context in any basic text book on the subject. For further information concerning the recalled topics see the related references.

2.1 Graphs

Although we assume that most of the readers are already acquainted with the issues addressed in this section, we introduce now the main topics on graphs due to the key role they play all along this work. Both the outlined problems, and the proposed solutions are abstracted as graphs.

The information in this section is commonly known and can be found in many books on this topic, for example books in references from [6] and [33]. You can also see [13]. We will focus on the main features of the particular class of graphs concerned in this work.

A graph can be seen as a diagram consisting of a set of vertices, also called nodes, together with lines joining certain pairs of these vertices. For example, the vertices could represent airports, and the lines the flights connecting them. Graphs are mathematical abstractions of this kind of situations. More precisely,

Definition 2.1.1
A graph G is an ordered pair (V, E) consisting on a nonempty set of vertices V and a set of edges E. Elements in E are pairs of elements of V, not necessarily distinct, called the endpoints of the edge.

Graphs are so named because they can be represented graphically, and it is this graphical representation which helps us to understand many of their properties. Each vertex is indicated by a point, and each edge by a line joining the points which represent its endpoints. Figure 2.1 shows a collection of graphs. Their vertices are A, B, C, D, E, F, G, and H.

There is not a unique way of drawing a graph, as the relative positions of points representing vertices and lines representing edges have no significance. We shall, however, often draw a diagram of a graph and refer to it as the graph itself.

If vertex $v \in V$ is an endpoint of edge $e \in E$, then v is said to be *incident* on e, and e is incident on v. An edge is said to *join* its endpoints, and a vertex is *adjacent* to another vertex if they are joined by an edge. The notation $V(G)$ and $E(G)$, or V_G and E_G, will be used for the vertex and edge sets respectively in case that G is not the only graph in consideration.

When the endpoints of an edge are the same vertex, the edge is said to be a *loop*. When they are different, the edge is said to be a *proper edge*. A *multi-edge* is a collection of two or more edges having identical endpoints. A *simple graph* is a graph that has no loops or multi-edges. Notice that graph in Figure 2.1b is a simple graph, for it has no loops nor multi-edges. In this work, we shall only consider simple graphs.

A *directed edge* is an edge e, one of whose endpoints is designated as the *source vertex*,

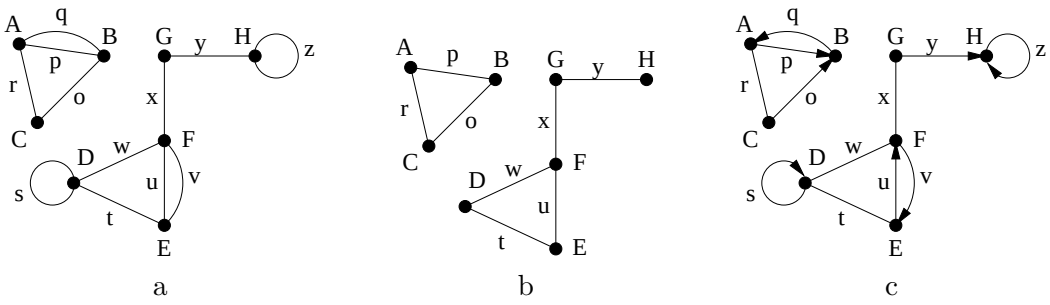


Figure 2.1: Different kinds of graph. a) Graph. b) Simple graph. c) Partially directed graph.

and whose other endpoint is designated as the *sink vertex*. They are denoted $source(e)$ and $sink(e)$, respectively. Directed edges are usually denoted by an arrow. Edges for which this distinction does not exist are called *undirected*. A *directed graph* is a graph whose edges are directed. Analogously, an *undirected graph* is a graph whose edges are undirected. A *partially directed graph*, is a graph that has undirected and directed edges. Graphs in Figures 2.1a and 2.1b are undirected and Figure 2.1c shows a partially directed graph, for some of its vertices are directed and some others not.

The *degree* of a vertex $v \in V$ in a graph G, denoted $\deg(v)$, is the number of proper edges incident on v plus twice the number of loops. For simple graphs, which are our subject of study, the degree is simply the number of adjacent vertices.

A graph H is a *subgraph* of G, written $H \subseteq G$, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. In this case, G is a *supra-graph* of H. Assuming that V' is a nonempty subset of V, the subgraph of G whose vertex set is V' and whose edge set is the set of those edges of G that have both ends in V' is called the subgraph of G *induced* by V' and is denoted by $G[V']$.

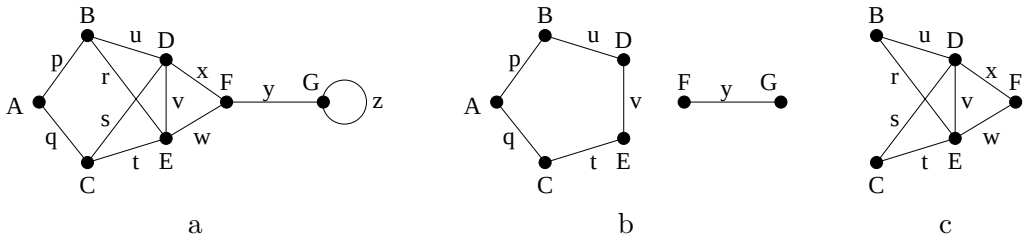


Figure 2.2: Subgraphs. a) Graph with vertices {A,B,C,D,E,F,G}. b) Subgraph of the graph depicted in a). c) Subgraph of the graph depicted in a) induced by the set of vertices V'={B,C,D,E,F}.

Figures 2.2b and 2.2c show two graphs which are subgraphs of the graph depicted in Figure 2.2a. Figure 2.2a is then a supra-graph of the graphs in Figures 2.2b and 2.2c. Figure 2.2c is also the subgraph induced by the set of vertices {B, C, D, E, F}.

2.1.1 Connection of graphs

A *walk* in G is a finite non-null sequence $W = v_0e_1v_1e_2v_2 \dots e_kv_k$, whose terms are alternately vertices and edges, such that, for $1 \leq i \leq k$, the ends of e_i are v_{i-1} and v_i. We say that W is a walk from v_0 to v_k. The integer k is the *length* of W. In a simple graph, a walk is determined by the sequence of its vertices. If the edges of a walk W are distinct and also the vertices are distinct, then W is called a *path*. We shall also use the word 'path' to denote a graph or subgraph whose vertices and edges are the terms of a path.

Two vertices u and v of G are said to be connected if there is a path in G from u to v. Connection is an equivalence relation on the vertex set V. Thus there is a partition of V into nonempty subsets $V_1, V_2, \dots, V_\omega$ such that two vertices u and v are connected if and only if both u and v belong to the same set V_i. The subgraphs $G[V_1], G[V_2] \dots, G[V_\omega]$ are called the *connected components* of G. If G has exactly one connected component, G is *connected*. Otherwise, G is *disconnected*. Notice that a path is connected if between every pair of vertices there is a path. The *distance* between two vertices in a graph is the length of the shortest path between them.

Graphs in Figure 2.1 are all three disconnected, with two connected components each. Graphs in Figures 2.2a and 2.2c are connected, and the one depicted in Figure 2.2b is disconnected.

2.1.2 The shortest path problem and the A* algorithm

With each edge e of G let there be associated a real number $w(e)$, called its *weight*. Then G, together with these weights on its edges, is called a *weighted graph*. In the airport graph example cited above, weights could represent the number of flights between each pair of airports. If H is a subgraph of a weighted graph, the weight $w(H)$ of H is the sum of weights on its edges. The *shortest path problem* consists on finding, in a weighted graph, a path of minimum weight connecting two specified vertices. The weight of a path is also called its *length*, and similarly the minimum weight of a path from u to v will be also called the *distance* between u and v in G.

A classic algorithm to solve the shortest path problem is the known as Dijkstra algorithm, [19], discovered by Dijkstra in 1959 and, independently, by Whitling and Hillier in 1960. For a complete review of the existing methods to solve the shortest path problem, see [93]. In this work we focus on the A* algorithm, [14], a more efficient method which

is complete and optimal under certain conditions. We describe the basics of this method following [91], where more information on this algorithm can be found.

The A* algorithm is based on the minimization of an evaluation function f which is actually the sum of two other functions:

$$f(n) = g(n) + h(n)$$

where $g(n)$ is the *path-cost* function and $h(n)$ an heuristic function. The $g(n)$ function gives the path cost from the starting node to the node n, and the $h(n)$ function is the estimated cost of the cheapest path from node n to the goal. Then, $f(n)$ is the estimate cost of the cheapest solution through n.

In order the A* algorithm to be complete and optimal, function $h(n)$ must never overestimate the real cost to reach the goal. Such an $h(n)$ is called an *admissible heuristic*. If $h(n)$ is admissible, then $f(n)$ never overestimates the actual cost of the best solution through n. A final observation is that among optimal algorithms of this type, A* is *optimally efficient*, that is, no other optimal algorithm is guaranteed to expand fewer nodes than A*. A proof of this result appears in [14].

2.2 Geometric constraint problems

In this work we model the problems we deal with as geometric constraint problems. A geometric constraint problem is made of a set of different geometric objects, related by a set of constraints among them. Many different approaches have been reported to solve the geometric constraint problem. In this section we formalize the notion of geometric constraint problem, analyze different solutions already known and describe thoroughly the one known as constructive.

2.2.1 Formal definition and properties

Geometric constraint solving is arguable a core technology of computer aided design and, by extension, geometric constraint solving is also applicable in virtual reality and is closely related in a technical sense to geometric theorem proving. For solution techniques, geometric constraint solving also borrows heavily from symbolic algebraic computation and matroid theory.

Many formulations of a geometric constraint problem have been given all along the literature, [9, 45, 48, 99, 106]. Following Hoffmann *et al.* geometric constraint problems can be categorized as either the general problem and the basic problem. The general problem can be characterized by means of a tuple $\Pi =< \Pi_E, \Pi_O, \Pi_X, \Pi_C >$ where

- Π_E is the geometric space constituting a reference framework into which the problem is embedded. Π_E is usually Euclidean.
- Π_O is the set of specific geometric objects which define the problem. They are chosen from a fixed repertoire including points, lines, circles and the like.
- Π_X is a, possibly empty, set of variables whose values must be determined. In general, variables represent quantities with geometric meaning: distances, angles and so on. When the quantities are without a geometric meaning, for example, when they quantify technological aspects and functional capabilities, those variables are called *external*.
- Π_C is the set of constraints. Constraints can be geometric or equational. Geometric constraints are relationships between geometric elements chosen from a predefined set, e.g., distance, angle, tangency, etc. The relationship (the distance, the angle, ...) is represented by a tag. If the tag represents a fixed value, known in advance, then the constraint is called *valuated*. If the tag represents a value to be computed as part of solving the constraint problem, then the constraint is called *symbolic*, [43].

  Equational constraints are equations some of whose variables are tags of symbolic constraints. The set of equational constraints can be empty.

The general geometric constraint solving problem can be now stated as follows:

Given a geometric constraint problem $\Pi =< \Pi_E, \Pi_O, \Pi_X, \Pi_C >$,

1. Are the geometric elements in Π_O placed with respect to each other in such a way that the constraints in Π_C and equations in Π_X are satisfied? If the answer is positive, then
2. Given an assignment of values to the valuated constraints and external variables, is there an actual construction that satisfies the constraints and equations?

When dealing with geometric constraint solving, the first issue that needs to be settled is the dimension of the embedding space Π_E. In 2D Euclidean space, $\Pi_E = \mathbb{R}^2$, a number of techniques have been developed that successfully solve the geometric constraint solving problem. For an in-depth review see Jermann, [60]. However, there remain open questions such as characterizing the competence (also called domain) of the known techniques.

Spatial constraint solving, where $\Pi_E = \mathbb{R}^3$, include problems in fields like molecular modeling, robotics, and terrain modeling. Here, both a good conceptualization and an effective solving methodology for the geometric constraint problem has proved to be difficult.

Pioneering work has been reported by Hoffmann and Vermeer, [49, 50] and by Durand, [20].

Presented in this way, the geometric constraint solving problem includes in general issues concerning how to deal with external variables. Here we refer the interested reader to the work by Hoffmann and Joan-Arinyo, [43], and Joan-Arinyo and Soto, [65].

The basic constraint problem only considers geometric elements and constraints whose tags are assigned a value. It excludes external variables, constraints whose tags must be computed, and equational constraints. So the basic problem is stated in the following way.

> Given a set Π_O with n geometric elements and a set Π_C with m geometric constraints defined on them
>
> 1. Is there a placement of the n geometric elements such that the m constraints are fulfilled? If the answer is positive,
> 2. Given an assignment of values to the m constraints tags, is there an actual construction of the n geometric elements satisfying the constraints?

In what follows we will focus on the basic geometric constraint solving problem.

Geometric constraint problems can be represented by a graph. The vertices shall represent the geometric elements of the problem and the edges shall represent the constraints among them. Given a geometric constraint problem $\Pi =< \Pi_G, \Pi_C, \Pi_P >$, the graph $G = (V, E)$ such that $V = \Pi_G$, $E = \Pi_C$ and the edges are labeled with the parameters Π_P represents the problem Π. The graph G is a simple graph, because constraints are defined upon two different geometric objects (thus, no loops are allowed) and between any two objects there is no more than one constraint (thus, no multi-edges are allowed). G is also undirected, since relations among geometric objects are non oriented.

Figure 2.3a shows an example of geometric constraint problem consisting of six points $\{a, b, c, d, e, f\}$ and nine point-point distance constraints $\{d_i, 1 \leq i \leq 9\}$ defined among them. Figure 2.3b shows the geometric constraint problem abstracted as a graph where each node represents one geometric element and each labeled edge represents a geometric constraint defined on the two geometric elements the edge connects. In what follows we shall represent geometric constraint problems as graphs.

Constraint solving community is mainly interested in objects which are invariant under rigid transformations of translation and rotation. This property is known as *rigidity*. Different kinds of rigidity have been defined, such as minimal or global rigidity, see [55, 58] and some works have been published describing operations which preserve it [68]. The intuitive concept of rigidity, the one that will be used in this work, is defined from

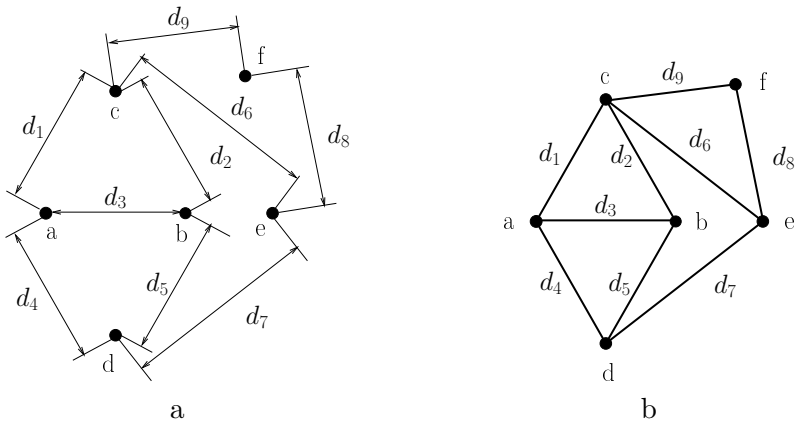


Figure 2.3: Geometric constraint problem example. a) Geometric sketch. b) Geometric constraint problem abstracted as a graph.

the number of solutions of the considered problem. In this context, geometric constraint problems are categorized in three different families:

1. Well-constrained problems are geometric constraint problems with a non-empty and finite set of solutions. In this work, we shall define rigid problems as well-constrained problems.

2. Over-constrained problems are those problems with no actual solution. Generally, the elimination of one or more constraints results in a well-constrained problem.

3. Under-constrained problems are geometric constraint problems for which an infinite set of solutions exists. Generally, the addition of one or more constraints results in a well-constrained problem.

Rigidity, as defined above, only refers to the problem's structure, and no other questions are considered. However, inconsistent situations in which specific assigned values given to the parameters result in a change of category may arise. Some works concerning this topic are for example from Laman, [75], and the more recent ones from Fudos and Hoffmann,[28], Hoffmann *et al.*, [43], Whiteley, [107], Henneberg [35] or Graver *et al.* [55].

As stated above, we identify well-constrained graphs with rigid graphs, that is, graphs which actually represent ruler-and-compass constructions. They are also known as Laman graphs after Gerard Laman, who first described them in 1970, [75]. Specifically, a graph $G = (V, E)$ is called Laman if $|V| \geq 3$ and G fulfills

1. $|E| = 2|V| - 3$,

2. For every subgraph $G' = (V', E')$ holds $|E'| \leq 2|V'| - 3$.

It is straightforward to see that Laman graphs have no disconnecting points and no vertices with degree zero nor one. An in depth discussion on Laman graphs will be presented in Chapter 7.

From now on, we will consider only well-constrained geometric constraint problems represented by undirected simple graphs, that is, graphs with no loops, no multi-edges and no direction established in their edges.

2.2.2 Constructive geometric constraint problems solving

Many techniques have been reported in the literature that provide powerful and efficient methods for solving geometric problems defined by constraints, which can be classified in three big groups: equational, based on the degree of freedom and constructive. For a complete review see [5, 45].

Equational methods are for example the numeric methods based on the Newton-Raphson algorithm [67], such as the systems described in [36, 78, 79, 82], or the algebraic symbolic methods which calculate the Gröbner basis of the equations system, like [12]. Methods based on the analysis of the degree of freedom are, among others, the works of Kramer, [71, 72, 73], or Hsu, [52, 53]. Among the constructive methods we find [8, 26, 27, 28], by Fudos *et al.* or [102], by Todd.

Among all the geometric constraint solving techniques, our interest here focuses on the one known as constructive. For an in depth discussion on this topic see, for example, [1, 9, 11, 28, 47, 48, 59, 64, 66, 84, 101] and the references there in. Computer programs that solve geometric problems defined by constraints are called *solvers*.

Constructive solvers yield the solution to the geometric problem defined by constraints as a sequence of construction steps that places each geometric element with respect to each other in such a way that the constraints are fulfilled. This sequence is called the *construction plan*. Construction plans represent a possibly exponential number of different solutions. In general, the construction plan that solves a constraint problem is not unique.

Figure 2.4 shows a construction plan for the constraint problem given in Figure 2.3. The meaning of each construction step is the usual. For example, $origin()$ stands for the origin of an arbitrary framework, $b = distD(a, d_3)$ places point b at distance d_3 from point a, $c_2 = circleCR(a, d_1)$ defines the circle c_2 with center a and radius d_1 and $intCC(c_1, c_2)$ defines a point as the intersection of circles c_1 and c_2. Notice that symbols c_i do not represent entities in the problem. They are intermediate results introduced to increase readability.

1.	a	$= origin()$	8.	d	$= intCC(c_4, c_5, s_2)$
2.	b	$= distD(a, d_3)$	9.	c_6	$= circleCR(c, d_6)$
3.	c_2	$= circleCR(a, d_1)$	10.	c_7	$= circleCR(d, d_7)$
4.	c_3	$= circleCR(b, d_2)$	11.	e	$= intCC(c_6, c_7, s_3)$
5.	c	$= intCC(c_2, c_3, s_1)$	12.	c_8	$= circleCR(c, d_9)$
6.	c_4	$= circleCR(a, d_4)$	13.	c_9	$= circleCR(e, d_8)$
7.	c_5	$= circleCR(d, d_5)$	14.	f	$= intCC(c_8, c_9, s_4)$

Figure 2.4: Construction plan for the example problem in Figure 2.3.

Constructive solvers are also known as decomposition-recombination planners (DR-planners), [48], since they follow the following strategy: first, perform the decomposition of the problem at hand in a concrete way, then analyze the obtained decomposition and finally construct the solution by recombining the different parts.

We shall refer as *decomposition step* to the split of a graph G into three different subgraphs G_1, G_2, G_3, called *clusters*, in such a way that $G_1 \cup G_2 \cup G_3 = G$ and $G_1 \cap G_2 = \{h_1\}, G_1 \cap G_3 = \{h_2\}, G_2 \cap G_3 = \{h_3\}$. Figure 2.5 illustrates the situation. Shared geometric elements h_1, h_2, h_3 are called *hinges*. The set of three hinges of a tree-decomposition step shall be called *triple of hinges* or *hinge triple*. If the three clusters G_1, G_2, G_3 include two vertices and one edge each, we say that the decomposition step is a *basic step*. For a more formal rational on this topic see [28] and [64].

If a graph G can be decomposed by applying successively decomposition steps to each cluster until every subgraph contains only two vertices and one edge between them, we shall say that the graph G is *tree-decomposable* and that the successive decomposition steps are a *tree-decomposition* for G. We shall also say that the problem represented by the graph G is tree-decomposable by extension. Unfortunately, not all rigid graphs can be decomposed in such a way, and in those cases the method will fail. An in depth discussion

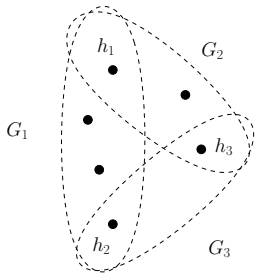


Figure 2.5: Sibling clusters pairwise share one geometric element.

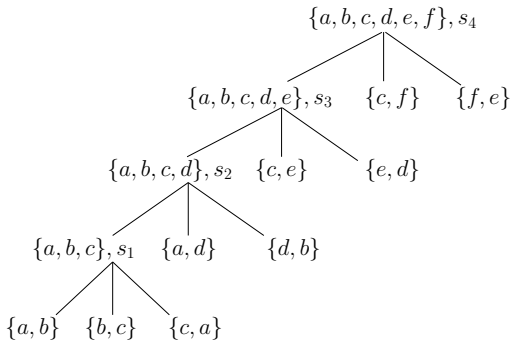


Figure 2.6: Construction plan as a tree decomposition.

on tree-decomposable graphs can be found in Chapter 7.

The tree-decomposition is a different and more convenient way to represent the construction plan, [66]. Figure 2.6 shows a decomposition tree for the construction plan in Figure 2.4. Notice that sibling clusters pairwise share one geometric element, for example clusters $\{a, b, c, d, e\}$, $\{f, e\}$ and $\{c, f\}$ pairwise share c, e and f respectively.

Once a graph has been decomposed, the tree-decomposition is used to construct a solution instance of the problem. Leaf nodes represent elemental placement problems corresponding to two geometric elements and the constraint defined on them. For example: two points at a given distance, a point and a straight segment at a given distance, two straight segments at a given angle and so on. The method starts by determining the position relative to each other of the two elements in a leaf node. Edges in the decomposition tree represent the combination of three solved clusters into a larger rigid cluster by application of a specific solving rule. Each node in the tree stands for a rigid object, built on the geometric objects included in the curly brackets list and whose position relative to each other has already been determined. The root node includes all the geometric elements in the problem and represents a solution instance.

In general, the tree-decomposition of a constraint problem is not unique. However, it has been proved that the tree-decomposition as defined above is canonical, [27, 64]. That means that the order in which the tree-decomposition is done is irrelevant, since they will always be the same decomposition steps. A consequence of that fact is that the hinge triples defined by the decomposition steps of a graph will always be the same, regardless of the concrete tree-decomposition at hand. That feature will give rise below to some interesting properties.

Solving a geometric constraint problem can be seen as solving a set of, in general, non linear equations. Therefore, each equation can have as many roots as the equation degree.

Obviously, each specific root will result in a different placement for the geometric elements in the problem. Selecting the desired root is known as the *root identification problem*, firstly addressed in [9]. A number of techniques have been developed to deal with the root identification problem. See, for example, [9, 61, 62, 104].

With each root we associate a *sign* which will characterize unequivocally the corresponding solution. We will call *index* to the set of all signs of a problem. The index in the construction plan in Figure 2.4 is $I = \{s_1, s_2, s_3, s_4\}$. For an in depth study of the index and the role it plays in geometric constraint solving see [24]. A similar definition can be found in [94]. The number of possible combinations of signs is bounded, as shown in [7].

The specific solution to the constraint problem Π identified by an assignment of values to the index I is called the *intended solution*. In what follows we consider that the intended solution has been fixed and that the degree of the equations underlying the geometric constraint problem is at most two, that is, signs s_i in the index take values in, say, $\{+, -\}$.

2.3 Problems with one variant parameter

When interacting with a computer featuring a mouse as an input device, mouse cursor position as it moves around the screen is captured in discrete steps. Therefore, intermediate positions are unknown. In dynamic geometry software, it is common practice to assume that the paths of free variables between two subsequent mouse events are linear, [70]. Thus, only one degree of freedom is left for the geometric element motion. In a more general framework, [15], the path is assumed to be polynomial in time t and the computation of the path itself is encoded leaving just one free variable t and in this way boiling down the problem to the situation with just one degree of freedom.

In this section we present basic concepts concerning geometric constraint problems for which the value of a given constraint parameter is not fixed, that is, geometric constraint problems with one variant parameter.

2.3.1 The construction plan as a function

In general, the concept of free geometric element in dynamic geometry can be captured in constructive geometric constraint-based dynamic geometry by considering the value assigned to a given constraint as a variable value. As we will see in Section 2.4.2, this does not have an effect on the constraint solving process and all what is needed is to reevaluate the construction plan as many times as needed.

To introduce the concept of movement in geometric constraint problems, we define geometric constraint problems with one variant parameter. Figure 2.7 illustrates the following

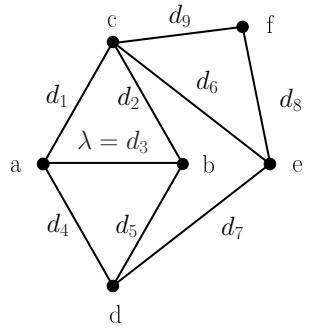


Figure 2.7: Geometric constraint problem with one variant parameter λ.

definition.

Definition 2.3.1
A geometric constraint problem with one variant parameter $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ *is a well-constrained geometric constraint problem such that all parameters in* Π_P *have been assigned a given value except for one, say* λ*, which can take arbitrary values in* $\mathbb{R}$*.*

The variant parameter may represent either a distance or an angle. We shall consider always positive distances, and angles defined inside the interval $[-\pi/2, \pi/2]$. Angles not included in this interval shall be wrapped to it modulo π.

Let T be a construction plan which solves the constraint problem $\Pi =< \Pi_G, \Pi_C, \Pi_P >$. Construction plans depend on the set of constraint parameters and on the index and they are valid for any problem derived from Π by considering one of its parameters as variant. Therefore, we can define the function construction plan as follows.

Definition 2.3.2
Let $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ *be a geometric constraint problem and* T *a construction plan for* Π*. Then,* $\mathrm{T}(\sigma, \lambda)$ *represents the evaluation of the construction plan* T *for the index assignment* σ *and a value* λ *of the variant parameter.*

Figure 2.8 shows from left to right objects in the family defined by the problem in Figure 2.7 for index value $\sigma = \{s_1 = +, s_2 = +, s_3 = +, s_4 = +\}$, distance constraint values $d_1 = 3, d_2 = 3, d_4 = 3.5, d_5 = 3.5, d_6 = 4, d_7 = 4.5, d_8 = 4, d_9 = 3.5$ and values of the variant parameter λ in {2.5, 4.5, 5.9}. That is, $\mathrm{T}(\sigma, 2.5)$, $\mathrm{T}(\sigma, 4.5)$ and $\mathrm{T}(\sigma, 5.9)$.

For some values of the variant parameter λ, however, it may not be possible to satisfy the set of constraints in Π_C, that is the construction plan T is unfeasible for such variant parameter values. The failure to instantiate the model poses naturally the question of how to compute ranges for parameters such that model instantiation is feasible. This problem

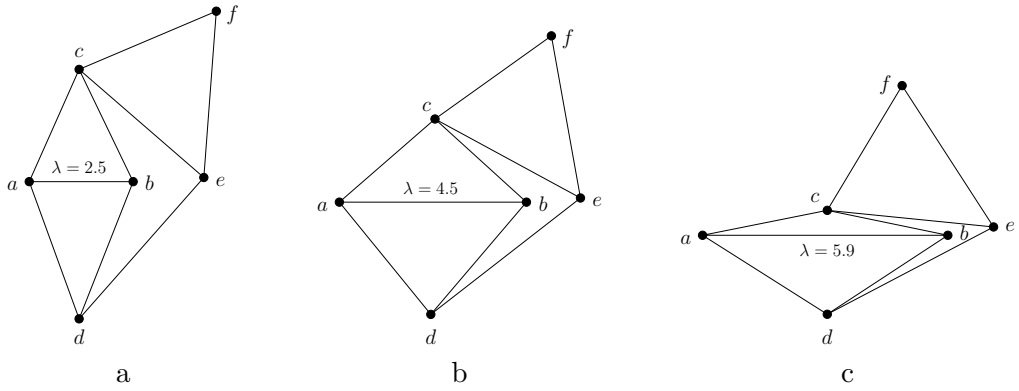


Figure 2.8: Objects belonging to the family defined by the problem in Figure 2.7. From left to right, $T(\sigma, 2.5)$, $T(\sigma, 4.5)$ and $T(\sigma, 5.9)$.

or restricted versions of it have been addressed in the literature.

Shapiro and Vossler, [96], and Raghothama and Shapiro, [86, 87, 88], developed a theory on validity of parametric family of solids by investigating the relationship between Brep and CSG schemas in systems with dual representations for solid modeling. The formulation is built on formalisms of algebraic topology. Unfortunately, it seems a rather difficult problem transforming these formalisms into effective algorithms.

Joan-Arinyo and Mata [63] reported on a method to compute feasible ranges for parameters in geometric constraint solving under the assumption that values assigned to parameters are non-trivial-width intervals. The method applies to complex systems of geometric constraints in both 2D and 3D and has been successfully applied in the dynamic geometry field, [25]. It is a general method, the main drawback, however, is that it is based on numerical sampling.

Hoffmann and Kim [46] developed a constructive approach to calculate parameter ranges for systems of geometric constraints that include sets of isothetic line segments and distance constraints between them. Model instantiation for distance parameters within the ranges output by the method preserve the topology of the set of isothetic lines.

In an illuminating work, van der Meiden and Bronsvoort, [105], reported on a constructive method to calculate parameter ranges for systems of geometric constraints. Constraint systems are restricted to systems of distance and angle constraints on points in 2D or 3D spaces that are well-constrained and decomposable into triangular and tetrahedral subproblems. The method automatically determines the allowable range for a single parameter of the system, called *variant parameter*, such that an actual solution exists for any value in the range. The van der Meiden method is one of the subjects of our study. In Chapter 4

1. $a = origin()$
2. $b = distD(a, d_1)$
3. $c_1 = circleCR(b, d_2)$
4. $l = linePA(a, \lambda)$
5. $c = intCL(c, l, s)$

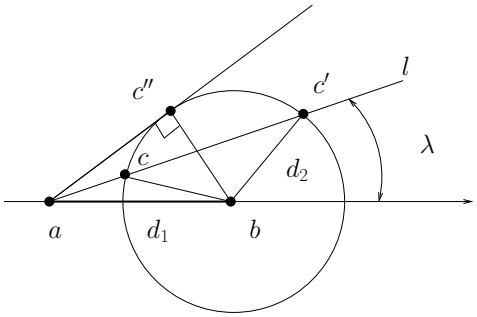


Figure 2.9: Critical values for a triangle defined by two sides and the angle supported by one of them. Construction plan and actual construction.

we shall prove that it is correct and complete, for it will be at the core of our approach to solve the reachability problem for geometric constraint based dynamic geometry.

Gao and Sitharam, in [29], described a general result concerning the computation of critical values for 2D problems with one degree of freedom which include just distance constraints and such that can be abstracted as one degree of freedom Henneberg graphs. Here we consider problems including distance and angle constraints such that can be abstracted as tree decomposable graphs, a superset of Henneberg graphs.

To formalize concepts related to construction plan feasibility, we call *critical variant parameter value*, or simply *critical value*, to the values λ_c of the variant parameter for which the feasibility of T changes. For the same critical value λ_c, a set of different constructions can be made depending on the chosen index.

To illustrate critical values, consider the construction shown in Figure 2.9 where a triangle is defined by giving the constraints $b = distD(a, d_1)$, $c = distD(b, d_2)$, and $\lambda = angle(ab, ac)$. If we assume that $d_1 \geq d_2$ and consider λ as the variant parameter, the construction plan shown on the left of Figure 2.9 is feasible for values of λ in the range $[0, \sin^{-1}(d_2/d_1)]$. The bounds of this range are the critical values of λ for this construction.

The situation described can be found for each basic construction in a constructive solver and the corresponding feasibility ranges can be collected in a dictionary. Table 2.1 shows examples for some basic constructions.

In this situation, we define the *domain* of λ as the set of values for which T is feasible. In general the domain of a variant parameter is a set of disjoint intervals bounded by critical variant parameter values. For a more formal definition of these concepts, see Chapter 4.

In the context that a construction plan is considered a function of the variant parameter, and considering that construction plan feasibility changes only at critical values, the solution instance generated by T(λ) traces a path in the space of solutions to problem Π

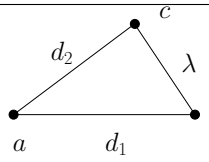
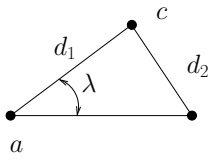
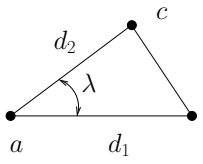
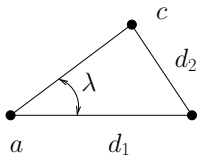
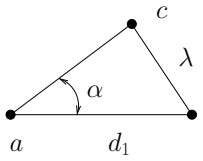
Basic Construction	Feasibility	
	$\text{abs}(\lvert d_1\rvert - \lvert d_2\rvert) \leq \lvert\lambda\rvert \leq \lvert d_1\rvert + \lvert d_2\rvert$	
	$-d_2/d_1 \leq \tan(\lambda) \leq d_2/d_1$	
	$0 \leq \lambda \leq 2\pi$	
	$-\sin(\lambda) \leq d_2/d_1 \leq \sin(\lambda)$	$d_1 \geq d_2$
	$0 \leq \lambda \leq 2\pi$	$d_1 < d_2$
	$d_1 \sin(\alpha) \leq \lambda \leq \infty$	

Table 2.1: Feasibility conditions for some basic construction steps. λ is the variant parameter.

as the value of λ changes continuously in its domain.

2.4 Dynamic geometry

Dynamic geometry appeared in the 80's, together with a number of software programs, as *Juno*, [82], which simulated in a computer the geometric ruler and compass constructions on paper. Two of the most relevant programs were *Cabri Geometry* [4, 74] and the *Geometer's Sketchpad* [56, 57], which are counted among the first dynamic geometry Systems.

The main feature of dynamic geometry is the dynamic character of the constructions. The system is able to record the way in which the user makes the construction, and can therefore redo it each time the user changes the value of a parameter or the position of a geometric element. Although the original purpose was merely constructive, the possibility of interacting with the construction and see how it changes in real time gave these kind of programs the notoriety they have nowadays.

Dynamic geometry systems are widely used in secondary schools for the teaching of geometry and mathematics, as they provide an intuitive and very accurate way of visualizing geometric objects and the relations between them. They are also multidisciplinary systems, since they can be used not only to represent geometric objects but also to represent graphs, functions, visualize transformations and introduce students into theorem proving and mathematical reasoning.

In this section we recall some of the basics in dynamic geometry, and present an architecture for a dynamic geometry system based on geometric constraint solving. A prototype with this architecture has been developed by Freixas *et al.*, [25], which will be the framework on top of which we build our work.

2.4.1 Basic concepts on dynamic geometry

A number of dynamic geometry systems have been reported in the literature. Besides those cited above, *Cinderella* [70, 89], developed by U. Kortenkamp and J. Richter-Gebert, or *GeoGebra* [31], have achieved an outstanding success due to its portability and the *wiki* associated to *GeoGebra*, [30], which provides teachers with lots of material for the teaching of geometry and mathematics.

Although every system is different to the others, and some of them have particular features that the others have not, as pointed out by Hözl, [51], all dynamic geometry systems share the following properties:

- they simulate ruler and compass constructions following Euclid's *Elements*.

1. p_1 = FREE
2. l_0 = JOIN((0,0), p_1)
3. c_1 = CIRCLE((0,0), 10)
4. c_2 = CIRCLE(p_1, 11)
5. p_2 = MEET(c_l, c_2)
6. l_1 = JOIN((0,0), p_2)
7. c_3 = CIRCLE(p_1, 15)
8. p_3 = MEET(l_1, c_3)

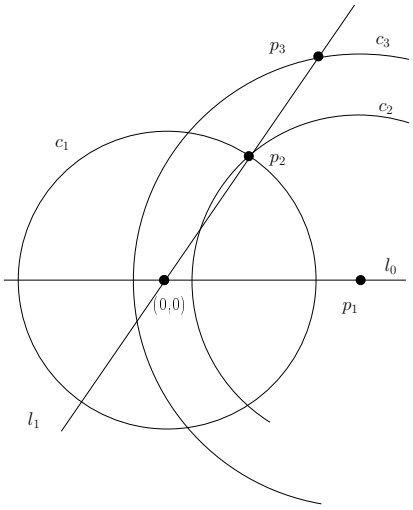


Figure 2.10: Example of GSP. Left) A GSP. Right) A GSP instance.

- they support macros simplifying the repetition of construction series, which can be defined by the user.
- they allow the user to move some parts of the construction without changing the underlying geometric constraints.

Different representations of geometric constructions in dynamic geometry have been proposed in the literature, [70, 83]. According to Kortenkamp and Denner-Broser, [16, 70], a convenient way to represent geometric constructions in dynamic geometry is Geometric Straight-Line Programs (GSP). A GSP consists of free points and dependent elements like a line through two points, the point where two lines intersect or the bisector of a segment. A GSP can be seen as a sequence of construction steps such that once values have been assigned to the free points, generates an actual construction that places free and dependent elements with respect to each others. Figure 2.10, shows a GSP and an actual construction, [15, 16].

The most important problem dynamic geometry must face is derived from ambiguities. Many constructions have not a unique solution (think for example in the intersection of a line and a circle), and to decide which one is the correct one, or at least the one the user is expecting to see, is not a simple question. Some characteristics, which we now explain, have been stated to define the behavior of the systems with respect to ambiguity, see for example [70].

Determinism is the property by which, in a dynamic geometry system, when performing the same movements from the same starting instance the system yields always the same

motion of the dependent elements. Determinism assures that, when passing through a point of multiple solutions, the system will always show the same one.

Conservatism in a dynamic geometry system guarantees that the final position of the dependent objects are the same for the same final position of the free objects, regardless of the path followed by them. That means that the final solution instance is independent from the path followed to reach it.

Finally, *continuity* is the property by which the movements of the dependent objects are continuous for continuous movements of the free elements. Continuity assures that no undesired "jumps" occur in the position of the geometric objects in the construction, and is a very desirable property, since users are expecting to see always a continuous behavior. Many works state that continuity and determinism are mutually exclusive, [16, 70, 83]. We will elaborate on this point in Chapter 6.

The main difference arising between geometric constraint solving and dynamic geometry is that, in dynamic geometry, the user is in charge of actually defining step by step the construction process that eventually will lead to the solution of the problem under study.

Setting up a problem in geometric constraint solving entails the geometric sketching of the problem by means of a user interface in which the set of geometric objects and relations among them are established. The solver analyzes then the problem, yielding the construction plan to solve it, if possible. Finally, a solution instance is shown on the screen, and the user is able to test the behavior of the construction by changing the specific parameter assignment given to the variant parameter.

Setting up a problem in dynamic geometry also entails the geometric modeling of the problem by means of a user interface, but the construction plan is actually given by the user when sketching the problem. Therefore a dynamic geometry system usefulness is basically limited by the user's abilities. The solution instance is shown on the screen, and the user is able to generate the motion of the whole construction by dragging a free geometric object in the model.

Table 2.2 summarizes the different actions necessary to set up a problem in geometric constraint solving and in dynamic geometry systems, specifying the actor of each action.

2.4.2 Constraint-based dynamic geometry

Constructive geometric constraint solving provides tools specifically well suited to support dynamic geometry systems. In particular, the construction plan of the problem at hand is stored, allowing to re-construct it for each possible value given to the variant parameter. Also, the index identifies uniquely each solution instance among the set of possible solution instances for a variant parameter value.

Action	Actor	
	Dynamic geometry	Geometric constraint
Geometric modeling	user	user
Construction plan	user	system
Testing	user	system/user

Table 2.2: Actions necessary to set up a problem, and their actors.

Although traditionally static, geometric constraint problems represent suitably dynamic geometry behaviors when we let a degree of freedom to move the system. In this way, dynamic geometry can be parameterized by the degree of freedom of the geometric constraint problem.

As pointed out above, the main drawback of dynamic geometry with respect to geometric constraint solving is the necessity of the intervention of the user in the construction of any geometric instance. Geometric constraint solving skips this problem thanks to the solver, which computes the placement of each geometric object observing the constraints among them.

We call constraint-based dynamic geometry to the inclusion of a constructive geometric constraint solver into a dynamic geometry system. Thanks to the constructive solver, constraint-based dynamic geometry is able to construct a solution instance without the need of the user, settling the problem above. Moreover, for any value given to the variant parameter, the system is able to compute the solution instance following the construction plan yield by the solver, and show the result in real time.

Setting up a problem in constraint-based dynamic geometry entails the geometric sketching of the problem by means of the dynamic geometry system user interface, specifying geometric objects and relations among them. The solver analyzes then the problem, yielding the construction plan to solve it, if possible. Finally, a solution instance is shown on the screen, and the user is able to test the behavior of the construction by changing the specific parameter assignment given to the variant parameter.

In [25], Freixas *et al.* reported on a Constraint-Based dynamic geometry System based on constructive geometric constraint solving. In this technology, the user defines a geometric problem by sketching some geometric elements taken from a given repertoire (points, lines, circles, etc) and annotates the sketch with a set of geometric relationships (point-point distance, point-line distance, angle between two lines and so on) that must be fulfilled.

Constructions in dynamic geometry can be easily transformed into Geometric Constraint Problems simply by expressing the relations existing among the geometric objects as constraints. Assume that the problem in Figure 2.10 has been defined at a Dynamic

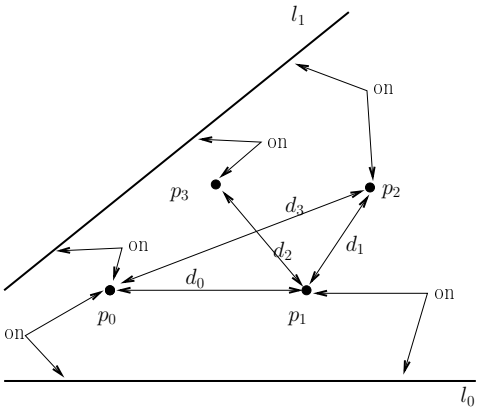


Figure 2.11: Geometric problem in Figure 2.10 expressed as a geometric constraint solving problem.

Geometry System interface, and that the set of geometric elements includes four points and two straight lines, $G = \{p_0, p_1, p_2, p_3, l_0, l_1\}$. Then, Figure 2.11 shows an equivalent way of defining the same problem G in a constraint based geometric system where d_0, d_1 and d_2 denote point-point distances and *on* denotes incidence.

The architecture for constructive solvers in which [25] is based is illustrated in Figure 2.12. Square boxes are *functional units* and rounded boxes are *data entities*. The functional units include the analyzer, the index selector and the constructor. The data entities are the geometric constraint problem, the construction plan, the parameters assignment and the index assignment.

Given a geometric constraint problem, $\Pi =< \Pi_G, \Pi_C, \Pi_P >$, the analyzer is responsible for figuring out whether the solver is able to solve the problem up to degenerated configurations, that is, whether it can find a description placement for the geometric objects. If the answer is positive, the analyzer outputs the construction plan, that will place the geometric elements in the position such that the constraints hold.

Selecting the desired root, which implies to solve the root identification problem, defined in Section 2.2.2, is the goal of the index selector that associates with each equation with several roots an index that unambiguously identifies the desired root. The index selection can be changed by the user at the user interface.

Finally, once a set of actual values have been assigned to the constraint parameters and the intended solution has been selected by assigning values to the index signs, the constructor builds an instance of a placement for the geometric objects, a solution instance, provided that no numerical incompatibility arises due to geometric degeneracy.

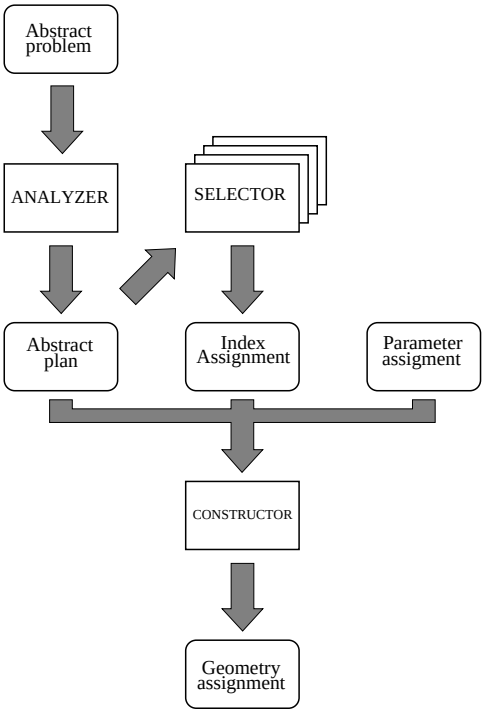


Figure 2.12: An architecture for the constructive solving technique.

We will consider the system in [25] as a basis on top of which we will build our approaches to solve the reachability and tracing problems in Geometric Constraint Based dynamic geometry. This architecture, known generically as DR-planner, [48], shows some nice properties.

First, the nature of the computations in each step is quite different. The analyzer requires symbolic computation while the constructor only performs numerical computations.

Second, determining whether the problem is solvable by the solver at hand or not is performed in the analysis step and it does not depend neither on the actual parameter values nor on the geometric computations.

Next, with the proposed decoupling, when computing instances for different parameter values, only the construction step needs to be carried out. This allows to skip the analysis step, which is computationally the most expensive, as well as the index selection.

Finally, given a symbolically solvable geometric constraint problem and a parameters assignment, the object can be instantiated if there are not numerical impossibilities, for example trying to intersect two disjoint circles that entails computing the square root of a negative value. These impossibilities are detected while carrying out the geometric computations, and we say that the construction plan is unfeasible.

CHAPTER 3

The hinges graph

We are all dependent on one another,
every soul of us on earth.

George Bernard Shaw

Construction plans of tree-decomposable graphs expressed by means of tree-decompositions are used to construct the solution instances of the geometric constraint problems associated to the graphs. However, tree-decompositions give no information about the internal structure of the graph at hand.

In this chapter we present a new representation for tree-decomposable Laman graphs which captures the intrinsic relations established between the different tree-decomposition steps of the associated geometric problem. The representation is based on hyper-graphs and captures additional useful properties of tree-decomposable graphs.

3.1 Dependency between tree-decomposition steps

In Chapter 2, tree-decomposable Laman graphs are characterized by having a tree-decomposition, represented as a tree. Tree-decompositions are made of tree-decomposition steps,

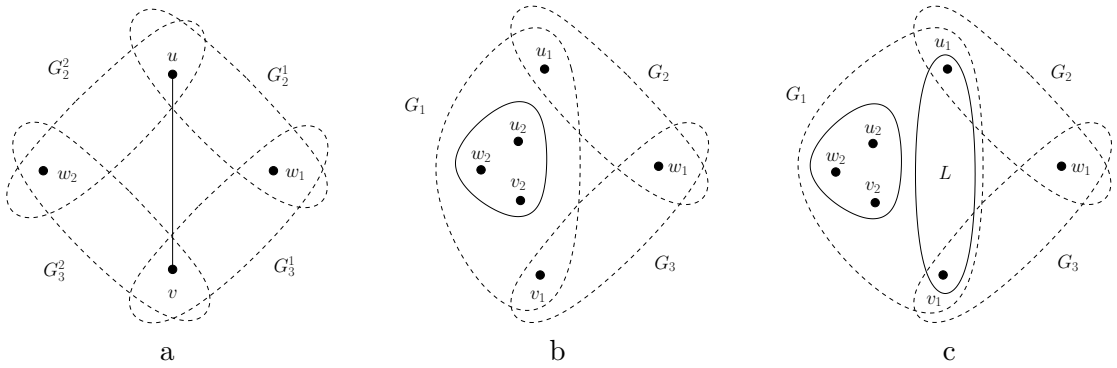


Figure 3.1: Dependence. a) Scheme of two directly dependent problems. b) Scheme of two indirectly dependent problems. In this case, problem with hinges (u_1, v_1, w_1) depends indirectly on the problem with hinges (u_2, v_2, w_2). c) Scheme of two independent problems.

in which the graph is split in three subgraphs called clusters by means of the hinge triple. The canonicity of the so defined tree-decomposition as stated by [27, 64] implies that a fixed tree-decomposition step in two different tree-decompositions of the same graph may split different subgraphs. The generation step is then not represented by the subgraph it splits or the clusters it creates, but by its hinge triple.

Consider a tree-decomposable Laman graph $G = (V, E)$, and a tree-decomposition. As said in Section 2.2.2, each tree-decomposition step identifies a set of three elements, called hinges, to which we refer as hinge triple. The hinge triple unequivocally defines the tree-decomposition step. From now on, we shall identify a tree-decomposition step by its hinge triple, and we shall represent this identification by bracketing the hinges.

In the process of decomposing a tree-decomposable Laman graph, some hierarchical relations among tree-decomposition steps may appear. As hierarchical level we must understand that necessary relations of priority are defined between the tree-decomposition steps, forcing the higher levels to be generated after the generation of the steps in lower levels. That happens, for example, when the construction of a specific tree-decomposition step must be performed necessarily before the construction of a different tree-decomposition step. We call this hierarchical relations *dependences*. We will distinguish two different kinds of dependence: direct and indirect. The direct dependence is defined as follows.

Definition 3.1.1
Let $G = (V, E)$ be a tree-decomposable Laman graph and T be a tree-decomposition of G. Let T_i, T_j be two different tree-decomposition steps in T with hinge triples (u, v, w_1) and (u, v, w_2) respectively. We say that T_i and T_j depend directly on each other if the edge $e = (u, v) \in E$.

We will consider from now on that $\mathcal{E}$ is the set of graphs $G = (V, E)$ such that $|V| = 2$ and $|E| = 1$, that is, the graph made of a unique edge, also called *edge graph*. We will denote as $G_{(a,b)}$ the graph $(\{a, b\}, \{(a, b)\}) \in \mathcal{E}$.

Figure 3.1a shows two directly dependent tree-decomposition steps. Tree-decomposition step T_1 merges clusters G_2^1, G_3^1 and $G_{(u,v)} \in \mathcal{E}$ by means of hinges u, v and w_1. Tree-decomposition step T_2 merges clusters G_2^2, G_3^2 and $G_{(u,v)} \in \mathcal{E}$ by means of hinges u, v and w_2. Both tree-decomposition steps share two of their hinges, and the edge joining them belongs to the graph. Then, the two generation steps depend directly on each other.

In plain words, direct dependence occurs when two tree-decomposition steps share a cluster with just one edge. That fact assures that the construction of one of the tree-decomposition steps is straightforward once the other has been already done. Notice that the direct dependence is symmetric, that is, if a tree-decomposition step T_i depends directly on the tree-decomposition step T_j, then also T_j depends directly on T_i. Thus, direct dependence fixes which tree-decomposition steps lie in the same hierarchical level.

The indirect dependence of two tree-decomposition steps is defined as follows.

Definition 3.1.2
Let $G = (V, E)$ be a tree-decomposable Laman graph and T be a tree-decomposition of G. Let T_i, T_j be two different tree-decomposition steps in T with hinge triples (u_1, v_1, w_1) and (u_2, v_2, w_2) respectively. Let G_1 be the cluster merged by T_i including u_1, v_1. We say that T_i depends indirectly on T_j if $u_2, v_2, w_2 \in V(G_1)$ and there is no tree-decomposable Laman subgraph L in G_1 such that $V(L)$ contains u_1, v_1 but not u_2, v_2, w_2.

Indirect dependence states the hierarchical relations between the tree-decomposition steps of a tree-decomposition of a graph. It represents the fact that for the proper building of a tree-decomposition step, the previous construction of other steps must have been already made. Figure 3.1b shows an scheme of two indirectly dependent problems. Cluster G_1 of the tree-decomposition step T_i defined by hinges u_1, v_1, w_1 includes the three hinges u_2, v_2, w_2 of the tree-decomposition step T_j. Figure 3.1c shows the case in which a tree-decomposable Laman subgraph L like the one described in Definition 3.1.2 exists.

Notice that the indirect dependence is asymmetric, that is, if a tree-decomposition step T_i depends indirectly on the tree-decomposition step T_j, then it is impossible that T_j depends indirectly on T_i. In particular, if G_1 is the cluster merged by T_i including u_1, v_1 and including also the hinges of T_j, $u_2, v_2, w_2 \in V(G_1)$, it is impossible that the merging vertex w_1 of T_i is included in any of the three clusters merged by T_j.

Indirect dependence is also transitive: if T_i depends indirectly on T_j, and T_j depends indirectly on T_k, then also T_i depends indirectly on T_k. The hinges of T_k, by definition, must be included in one of the clusters merged by T_j. The hinges of T_j are included in one of the clusters merged by T_i, say G_1, and then, all the clusters merged by T_j are included

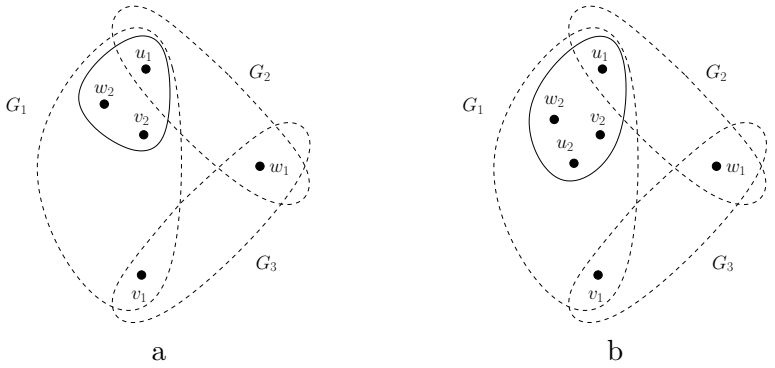


Figure 3.2: Strong dependence. a) Strong dependence in the case that T_2 contains a hinge of T_1. b) Strong dependence in the case that a tree-decomposition step T_3 in which T_2 depends indirectly contains a hinge of T_1.

in G_1. Then, G_1 includes the merging vertices of T_k.

Let $T_i = (u, v, w)$ be a tree-decomposition step of a tree-decomposable graph $G = (V, E)$ and $(u, v), (u, w), (v, w) \in E$. Then T_i has no indirect dependences, and it is called a *basic tree-decomposition step*. If at least one of the edges $(u, v), (u, w), (v, w)$ is not in E, T_i has necessarily some indirect dependences and is called a *merge tree-decomposition step*.

We consider a particular case of indirect dependency, which we call *strong dependency*, defined as follows. Refer to Figure 3.2.

Definition 3.1.3
Let $G = (V, E)$ be a tree-decomposable Laman graph and T be a tree-decomposition of G. Let T_i, T_j be two different tree-decomposition steps in T with hinge triples (u_1, v_1, w_1) and (u_2, v_2, w_2) respectively such that T_i depends indirectly on T_j. We say that T_i depends strongly on T_j if

1. *either $u_1 = u_2$, or there exists a tree-decomposition step T_k with hinges (u_3, v_3, w_3) such that T_j depends indirectly on T_k and $u_1 = u_3$, and*
2. *there is no tree-decomposition step T_l such that T_i depends indirectly on T_l and T_l depends indirectly on T_j.*

For each cluster, there exist at least one and up to two different tree-decomposition steps in which T strongly depends. We will explain this point in detail in Section 3.5.

It may also happen that no dependence is defined between some pairs of problems. We say that two tree-decomposition steps are independent if they are not related by a direct nor

an indirect dependence. The formal definition of independence between tree-decomposition steps, illustrated in Figure 3.1c, is the following one.

Definition 3.1.4
Let $G = (V, E)$ be a tree-decomposable Laman graph and T be a tree-decomposition of G. Let T_i, T_j be two different tree-decomposition steps in T with hinge triples (u_1, v_1, w_1) and (u_2, v_2, w_2) respectively. Let G_1 be the cluster merged by T_i including u_1, v_1. We say that T_i and T_j are independents if T_i and T_j are not related by any direct nor indirect dependence.

3.2 Definition of the hinges graph

We define now a graph, which we will call the *hinges graph*, or h-graph in short, associated to a tree-decomposable Laman graph G. It represents the tree-decomposable graph G together with its tree-decomposition. The h-graph is a hyper-graph which captures dependences among the tree-decomposition steps of the tree-decomposition. Vertices represent tree-decomposition steps of a tree-decomposition of G, symbolized by the hinge triples into brackets. Edges represent relations of dependence among them. For the sake of readability, we will refer to vertices of the h-graph as *nodes*.

As seen in Section 3.1, indirect dependence is transitive, thus a h-graph specifying all relations of indirect dependence between the tree-decomposition steps of a tree-decomposable Laman graph would be unnecessarily complicated. We consider therefore only direct and strong dependences. We will show later on that strong dependence suffices to represent relations of indirect dependence between tree-decomposition steps.

Definition 3.2.1
Let $G = (V, E)$ be a tree-decomposable Laman graph. The hinges graph, or h-Graph, associated to G is the graph $\mathcal{H}(G) = (\mathcal{V}, E_D, E_S)$, where $\mathcal{V}$ is the set of hinge triples of G, E_D is the set of pairs (V_1, V_2) such that the tree-decomposition steps represented by nodes V_1, V_2 depend directly on each other, and E_S is the set of pairs (V_1, V_2) such that the tree-decomposition step represented by node V_2 strongly depends on the tree-decomposition step represented by node V_1.

Direct dependence is represented by non-directed edges which will be called d-edges. Strong dependency is represented by directed edges which will be called s-edges. Figure 3.3a shows a tree-decomposable Laman graph example G. Figure 3.3b shows the associated h-graph.

In an intuitive way, d-edges represent the fact that the two joined nodes are at the same hierarchical level, whereas s-edges represent that one of the nodes is at a higher hierarchical level than the other, that is, that one of the nodes must be constructed necessarily before

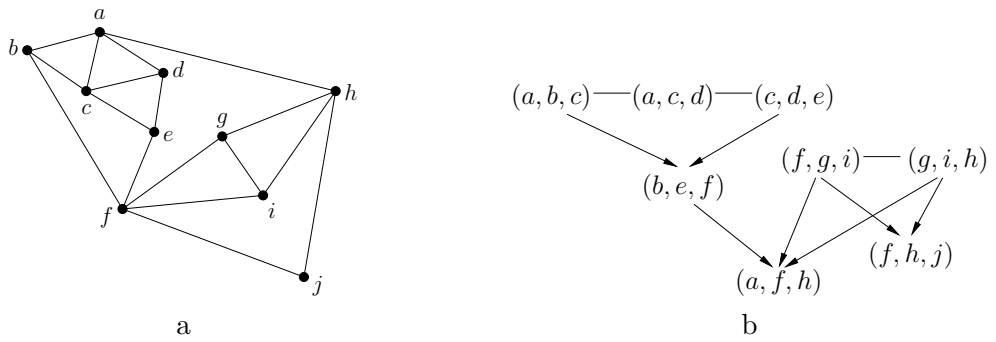


Figure 3.3: Example of h-graph. a) Tree-decomposable Laman graph G. b) h-graph $\mathcal{H}(G)$ associated to G.

the construction of the other.

Notice that in Figure 3.3a, tree-decomposition step (b, e, f) indirectly depends on tree-decomposition step (a, c, d). However, since the dependence is not strong, no s-edge from (a, c, d) to (b, e, f) exists in Figure 3.3b. We will explain in detail this point in Section 3.5.

According to what has been explained in Section 3.1, nodes to which no s-edge arrives correspond to basic steps and nodes to which a s-edge arrives, represent merge steps.

By the canonicity of the tree-decomposition stated in [27, 64], all possible tree-decompositions of G share the same tree-decomposition steps. Based on that fact, we will see that the h-graph associated to a tree-decomposable Laman graph is unique.

Theorem 3.2.1
Let G be a tree-decomposable Laman graph and $\mathcal{H}(G)$ the associated h-graph. Then $\mathcal{H}(G)$ is unique.

Proof
In [27, 64] it has been shown that every tree-decomposition of a tree-decomposable Laman graph G have the same set of tree-decomposition steps. That means that the set of hinge triples is fixed. Then, the set of nodes of $\mathcal{H}(G)$ does not depend on the specific tree-decomposition considered.

Also, edges are defined according to the relations of dependence arising between the different tree-decomposition steps in the graph. This reasoning shows that the hinges graph is unique. □

Notice that the unicity of the h-graph associated to a tree-decomposable graph assures that the h-graph is independent from any concrete tree-decomposition. We can establish an injective relation from the set of tree-decomposable graphs to the set of h-graphs, as

every tree-decomposable Laman graph has a different associated h-graph.

The hinges graph shows some other nice properties. For example, the h-graph can be interpreted as the graph G enhanced with the information about its tree-decomposition, it gives information about the structure of the graph and that it can be easily constructed from the construction plan of the graph. We justify these properties in the following sections. More features of h-graphs will be described in Section 7.

3.3 H-graph from a construction plan

In this section we analyze the intrinsic relation between construction plans and h-graphs, showing how to construct h-graphs from construction plans and explaining how to derive any possible construction plan from the h-graph associated to the problem at hand.

We present first a simple algorithm which computes the h-graph of a tree-decomposable Laman graph once it has already been tree-decomposed by a solver. Consider that we decompose the graph G into clusters G_1, G_2, G_3 by means of hinges u, v, w, such that $u, v \in V(G_1)$, $u, w \in V(G_2)$ and $v, w \in V(G_3)$. Assume that we know the h-graphs $\mathcal{H}(G_1)$, $\mathcal{H}(G_2)$ and $\mathcal{H}(G_3)$. Then, for each cluster G_i, and depending on whether the edge joining the two hinges in G_i is included in G, we compute the nodes of direct or strong dependence and join them to the new node (u, v, w).

Algorithm 1 shows the recursive algorithm which computes the h-graph of a tree-decomposable Laman graph from its construction plan. T represents the tree-decomposition, where each node stores the information of the elements included in each cluster and the sons it has as well as the three hinges h1, h2, h3 which decompose the step. HG represents a h-graph and stores the sets HV, ED and ES standing for the nodes set, the set of direct dependences and the set of strong dependences respectively. If HG_1 and HG_2 represent the h-graphs with sets HV_1, ED_1 and ES_1, and HV_2, ED_2 and ES_2 respectively, we indicate by $\mathrm{HG}_1 \cup \mathrm{HG}_2$ the unions $\mathrm{HV}_1 \cup \mathrm{HV}_2$, $\mathrm{ED}_1 \cup \mathrm{ED}_2$ and $\mathrm{ES}_1 \cup \mathrm{ES}_2$. The function *Compute_Strong_Dependences* is shown in Algorithm 2, Section 3.5.

We show now how to derive a tree-decomposition or construction plan for G from $\mathcal{H}(G)$. That will show that $\mathcal{H}(G)$ is a unique representation of all possible tree-decompositions of G.

In general, it is not possible to begin the tree-decomposition of a graph by an arbitrary hinge triple. We must find a triple which splits the graph into three clusters pairwise sharing one element. If a tree-decomposition step T_i depends indirectly on another step T_j, it is impossible that the hinges of T_j represent a tree-decomposition step of the graph, see Figure 3.1b. Then, the triples defining a tree-decomposable step of the graph are those on which no other node depends indirectly. These triples are easily determined in a h-

Algorithm 1 Computing the hinges graph from the tree decomposition

```
Input: T, the tree-decomposition of a graph G
Output: HG = ( HV, ED, ES ), the h-graph associated to G

function Compute_Node()
HG = ∅
V0 = ( T.h1, T.h2, T.h3 )
if Number of sons of T > 0 then
  for each T_i son of T do
    HG_i = Compute_Node( T_i )
    HG = HG ∪ HG_i
    if i == 0 then
      HG.HV = HG.HV ∪ { V0 }
    end if
    h1 := First hinge in HG_i and T.hinges
    h2 := Second hinge in HG_i and T.hinges
    if ( h1, h2 ) in E then
      D = Compute_Direct_Dependences( V0, HG_i )
      for each D_i in D do
        HG.ED = HG.ED ∪ { ( D_i, V0 ) }
      end for
    else
      S = Compute_Strong_Dependences( V0, HG_i )
      for each S_i in S do
        HG.ES = HG.ES ∪ { ( S_i, V0 ) }
      end for
    end if
  end for
  return HG
else
  HV = HV ∪ { V0 }
  return HG
end if
endfunction
```

graph. If no triple depends on node V, no s-edge will have V as source. Going back to the graph depicted in Figure 3.3a, notice that the only tree-decomposition steps for which a tree-decomposition step of the graph is possible are (a, f, h) and (f, h, j). In Figure 3.3b we can find the nodes representing these tree-decomposition steps simply by following the direction of the s-edges.

Once the first tree-decomposition step is chosen, we decompose the graph G by its hinges. The h-graph $\mathcal{H}(G)$ is decomposed in the same way just by removing the chosen triple and the s-edges having it as sink node. If the resulting $\mathcal{H}(G)$ has more than one connected component the same will occur in G. The decomposition is repeated then in each one of the resulting connected components recursively.

In Figure 3.3a, consider that the first chosen tree-decomposition step has $\{a, f, h\}$ as hinges. Graph G is split into three subgraphs one of which is in $\mathcal{E}$. The node (a, f, h) in the h-graph $\mathcal{H}(G)$ in Figure 3.3b is removed, as well as all the s-edges arriving to it. Then, the h-graph is split into two subgraphs, associated to each of the two clusters not in $\mathcal{E}$. The cluster in $\mathcal{E}$ has no associated h-graph.

As seen, in order to generate a construction plan from a h-graph, nodes in the h-graph must be decomposed following the dependences established among them, represented by the s-edges. However, in the case that the triples are related only by direct dependences, there is no priority in the order of tree-decomposition of the nodes. Each one of the choices in the order in which these tree-decomposition steps are decomposed will give rise to a different final tree-decomposition. In the subgraph made of vertices a, b, c, d, e in Figure 3.3a, the order of decomposition of the nodes $(a, b, c), (a, c, d), (c, d, e)$ can give rise to up to 6 different construction plans.

3.4 Subgraphs and complete subgraphs

In this section we describe the subgraphs $\mathcal{H}'(G)$ of a h-graph $\mathcal{H}(G)$, and analyze the conditions under which they are the associated h-graphs of tree-decomposable Laman subgraphs G' of the original graph G. In the first place we give a definition of h-graphs subgraphs, which will be analogous to the regular subgraph concept.

Definition 3.4.1
Let $G = (V, E)$ be a tree-decomposable Laman graph, and $\mathcal{H}(G)$ be the h-graph associated to G. The h-graph $\mathcal{H}'(G) = (\mathcal{V}', E'_D, E'_S)$ is a subgraph of $\mathcal{H}(G)$ if $\mathcal{V}' \subseteq \mathcal{V}$, $E'_D \subseteq E_D$ and $E'_S \subseteq E_S$.

Among the possible subgraphs of a h-graph, we are interested in those which are associated to a Laman graph. This type of subgraph is called a *complete* subgraph, and is defined as follows. We denote by $\text{dep}_{\mathcal{H}}(V)$ the set of nodes that depend on node V in the

h-graph $\mathcal{H}(G)$.

Definition 3.4.2
Let $G = (V, E)$ be a tree-decomposable Laman graph, and $\mathcal{H}(G)$ be the h-graph associated to G. Let $\mathcal{H}'(G) = (\mathcal{V}', E'_D, E'_S)$ be a subgraph of $\mathcal{H}(G)$. Then $\mathcal{H}'(G)$ is called complete if for all $V \in \mathcal{V}'$, $dep_{\mathcal{H}}(V) \subseteq \mathcal{V}'$.

Subgraphs $\mathcal{H}'(G)$ fulfilling this condition are called complete because they include all the dependences of every node in them. In other words, every tree-decomposition step in a complete subgraph can be constructed because the subgraph contains all the necessary nodes to perform the construction. Then, the whole subgraph can be constructed, generating a tree-decomposable Laman graph. In particular, the tree-decomposable Laman graph associated to a subgraph of a h-graph fulfills the following property.

Lemma 3.4.2
Let $G = (V, E)$ be a tree-decomposable Laman graph, and $\mathcal{H}(G)$ be the h-graph associated to G. Let $\mathcal{H}'(G)$ be a complete subgraph of $\mathcal{H}(G)$. Then there exists a tree-decomposable Laman subgraph $G' \subseteq G$ such that $\mathcal{H}'(G) = \mathcal{H}(G')$.

Proof
Since $\mathcal{H}'(G) = (\mathcal{V}', E'_D, E'_S)$ is complete, it is the h-graph associated to a Laman graph, say $G' = (V', E')$. We will prove that G' is a subgraph of G.

The set of vertices in V' are the elements in the nodes of $\mathcal{V}'$, which in turn are a subset of the nodes in $\mathcal{V}$. Those nodes are made with triples of vertices of V. Therefore, $V' \subseteq V$.

A h-graph defines unequivocally the edges which are included in the associated Laman graph, because each cluster different from an edge graph is represented by a subgraph. Consider an edge $e = (v_1, v_2) \in E'$. In [106], Vila showed that to each edge $e = (v_1, v_2)$ in G corresponds a leaf $\{v_1, v_2\}$ in the tree-decomposition of G, which means that every edge is a cluster in at least one tree-decomposition step. Then, there is at least one node V in $\mathcal{V}'$ including vertices v_1, v_2 as hinges. The existence of e implies that, in the tree-decomposition step V, no cluster has been defined upon vertices v_1, v_2. Since $\mathcal{V}'$ includes all the dependences of V in $\mathcal{H}(G)$, no cluster is defined upon vertices v_1, v_2 in $\mathcal{H}(G)$, and then $e \in E$. Therefore, $E' \subseteq E$. □

Lemma 3.4.2 tells that every complete subgraph of $\mathcal{H}(G)$ induces a tree-decomposable Laman subgraph in the graph G. Now we prove that inclusion is preserved when computing h-graphs.

Lemma 3.4.3
Let G_1 and G_2 be two tree-decomposable Laman graphs, and $\mathcal{H}(G_1), \mathcal{H}(G_2)$ the associated h-graphs, respectively. Then, $\mathcal{H}(G_1) \subseteq \mathcal{H}(G_2)$ if and only if $G_1 \subseteq G_2$.

Proof
For the only if part, apply the same proof from Lemma 3.4.2 for $G_1 = G'$ and $G_2 = G$.

For the if part, consider that $G_1 \subseteq G_2$. Then, for every tree-decomposition step of G_2 with hinges (u, v, w) and clusters C_1, C_2, C_3, either G_1 is included in one of the clusters, or (u, v, w) is a hinge triple of G_1. To show that, consider that G_1 is not included in any of the clusters C_1, C_2, C_3. Then there are elements of G_1 in at least two of the clusters, say C_1 and C_2, which share only one node. Since G_1 is Laman, there must be elements of G_1 also in C_3, otherwise G_1 would have a disconnecting point. Then, (u, v, w) is also a hinge triple of G_1.

Applying the same rational to every cluster yielded by the decomposition step (u, v, w), we conclude that every decomposition step of G_1 is included in the set of decomposition steps of G_2. By the canonicity of the tree-decomposition, there exists a tree-decomposition of G_2 such that G_1 is one of the clusters of a tree-decomposition step. By Algorithm 1, $\mathcal{H}(G_1) \subseteq \mathcal{H}(G_2)$. □

From now on, all the subgraphs of h-graphs considered in this work will be complete subgraphs.

3.5 Representative nodes of complete subgraphs

In this section we analyze the relation between strong dependency and complete subgraphs. In order to better explain how indirect dependence is represented in a h-graph, we introduce the concept of minimum complete subgraph of $\mathcal{H}(G)$ spanned by a set of nodes Q.

Definition 3.5.1
Let $G = (V, E)$ be a tree-decomposable Laman graph, and $\mathcal{H}(G) = (\mathcal{V}, E_D, E_S)$ be the h-graph associated to G. Let $Q \subset \mathcal{V}$ a subset of nodes. Then the minimum complete subgraph spanned by Q, $\mathcal{H}_Q(G)$, is defined as the minimum complete subgraph of $\mathcal{H}(G)$ including all the nodes in Q.

Notice that the minimum complete subgraph of $\mathcal{H}(G)$ spanned by a set of nodes is complete by definition, which means that it is the h-graph associated to a Laman subgraph of G.

An example is shown in Figure 3.4. Figure 3.4a shows the minimum subgraph spanned by nodes $(a, b, c), (b, e, f)$ from graph $\mathcal{H}(G)$ in Figure 3.3b. Figure 3.4b shows the tree-decomposable Laman graph associated to this h-graph, which clearly is a subgraph of the one represented in Figure 3.3a, and it is also complete. Figure 3.4a is also the minimum tree-decomposable Laman subgraph spanned just by node (b, e, f).

Every complete subgraph of $\mathcal{H}(G) = (\mathcal{V}, E_D, E_S)$ can be spanned by a set of its vertices,

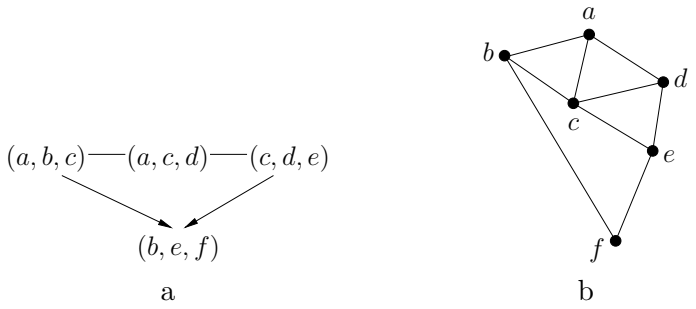


Figure 3.4: Complete subgraph. a) H-graph $\mathcal{H}(G')$ which is a subgraph of the h-graph depicted in Figure 3.3b. b) Tree-decomposable Laman subgraph G' associated to $\mathcal{H}(G')$, which is a subgraph of the graph depicted in Figure 3.3a.

since $\mathcal{H}(G) = \mathcal{H}_{\mathcal{V}}(G)$. In particular, since $\mathcal{H}(G)$ is also a complete subgraph of itself, every h-graph can be spanned by a set of its vertices. We call the minimum set of vertices which span $\mathcal{H}(G)$ the *representative nodes* of $\mathcal{H}(G)$. The formal definition is stated as follows. We define the distance between two nodes v_1, v_2 of a h-graph as the length of the shortest path connecting nodes v_1 and v_2.

Definition 3.5.2
Let $G = (V, E)$ be a tree-decomposable Laman graph, and $\mathcal{H}(G) = (\mathcal{V}, E_D, E_S)$ be the h-graph associated to G. The representative nodes of $\mathcal{H}(G)$ are the nodes in the set $Q \subset \mathcal{V}$ such that $\mathcal{H}_Q(G) = \mathcal{H}(G)$ and the sum of distances in $\mathcal{H}(G)$ between every pair of nodes in Q is minimum.

Going back to the example graph in Figure 3.4b, the representative vertex of the h-graph is (b, e, f). The distance between the nodes of a set with just one node is considered zero.

Once a set of nodes which spans a h-graph has been determined, the distance between a pair v_1, v_2 of them can be minimized by checking if the different nodes v_i in the path from v_1 to v_2 also span the h-graph. We will show now how to compute the nodes on which a tree-decomposition step strongly depends. First, we introduce the minimum h-graph subgraph including a set of elements of G.

Definition 3.5.3
Let $G = (V, E)$ be a tree-decomposable Laman graph, and $\mathcal{H}(G) = (\mathcal{V}, E_D, E_S)$ be the h-graph associated to G. The minimum complete subgraph spanned by the set $P \subset V$, $\mathcal{H}_P(G) = (\mathcal{V}_P, E_D^P, E_S^P)$, is the minimum complete subgraph of $\mathcal{H}(G)$ such that for each $v_i \in P$ there is a node $V_i \in \mathcal{V}_P$ with $v_i \in V_i$.

Since the minimum complete subgraph spanned by a set of vertices P is complete, it has

an associated tree-decomposable Laman graph. Assuming the notation as in the previous definition, we denote by G_P this graph. Using Lemma 3.4.2 we can easily check that G_P is the minimum tree-decomposable Laman subgraph of G containing all the elements in P.

Lemma 3.5.4
Let $G = (V, E)$ be a tree-decomposable Laman graph, and $\mathcal{H}(G) = (\mathcal{V}, E_D, E_S)$ be the h-graph associated to G. Let $\mathcal{H}_P(G)$ be the minimum complete subgraph spanned by the set $P \subset V$. Then, G_P is the minimum tree-decomposable Laman subgraph of G containing all the elements in P.

Proof
By Lemma 3.4.2, G_P is a tree-decomposable Laman subgraph of graph G. G_P contains all the elements in the nodes of $\mathcal{H}_P(G) = (\mathcal{V}_P, E_D^P, E_S^P)$, and by definition, for each $v_i \in P$ there exist a node $V_i \in \mathcal{V}_P$ with $v_i \in V_i$. Then, G_P contains all the elements in P.

Consider that G_P is not minimum. Then, there exists a tree-decomposable Laman subgraph G_0 of G containing all the elements in P and such that G_0 is also a subgraph of G_P. Then, by Lemma 3.4.3, $\mathcal{H}(G_0) \subseteq \mathcal{H}(G_P)$, which is a contradiction with the fact that $\mathcal{H}(G_P)$ is the minimum complete subgraph spanned by the set $P \subset V$. □

In the scope of this work, only subgraphs spanned by sets of two elements in V will be considered. Thus, if $P = \{u, v\}$, we shall denote $\mathcal{H}_P(G)$ as $\mathcal{H}_{u,v}(G)$ and G_P as $G_{u,v}$.

Consider a tree-decomposition step T_i of a graph G with hinges (u, v, w). We prove now that T_i depends indirectly on the tree-decomposition steps necessary to construct $\mathcal{H}_{u,v}(G)$, $\mathcal{H}_{u,w}(G)$ and $\mathcal{H}_{v,w}(G)$.

Theorem 3.5.5
Let $G = (V, E)$ be a tree-decomposable Laman graph and $\mathcal{H}(G)$ the associated h-graph. Let T be a tree-decomposition of G, and T_i a tree-decomposition step in T with hinges u, v, w. Let $\mathcal{H}_{u,v}(G)$ be the graph spanned by the two hinges u, v. If $(u, v) \notin E$, then T_i depends indirectly on the nodes in $\mathcal{H}_{u,v}(G)$ and is independent on the other ones.

Proof
Let G_1 be the tree-decomposable Laman cluster merged by the tree-decomposition step T_i such that $u, v \in V(G_1)$. Then, the associated h-graph $\mathcal{H}(G_1) = (\mathcal{V}_1, E_D^1, E_S^1)$ is complete, and as $u, v \in V(G_1)$, there exist at least two nodes $U, V \in \mathcal{V}_1$ such that $u \in U$ and $v \in V$. Since $\mathcal{H}_{u,v}(G)$ is the minimum h-graph fulfilling this last property, $\mathcal{H}_{u,v}(G) \subseteq \mathcal{H}(G_1)$. Consider the tree-decomposable Laman graph $G_{u,v}$ associated to the complete h-graph $\mathcal{H}_{u,v}(G)$. By Lemma 3.4.3, $G_{u,v} \subseteq G_1$, see Figure 3.5a.

Then, the hinges of every tree-decomposition step (u', v', w') in the tree-decomposition T leading to $G_{u,v}$ are included in G_1. Moreover, since $G_{u,v}$ is the minimum tree-decomposable Laman graph including u, v, there is no tree-decomposable Laman subgraph in $G_{u,v}$

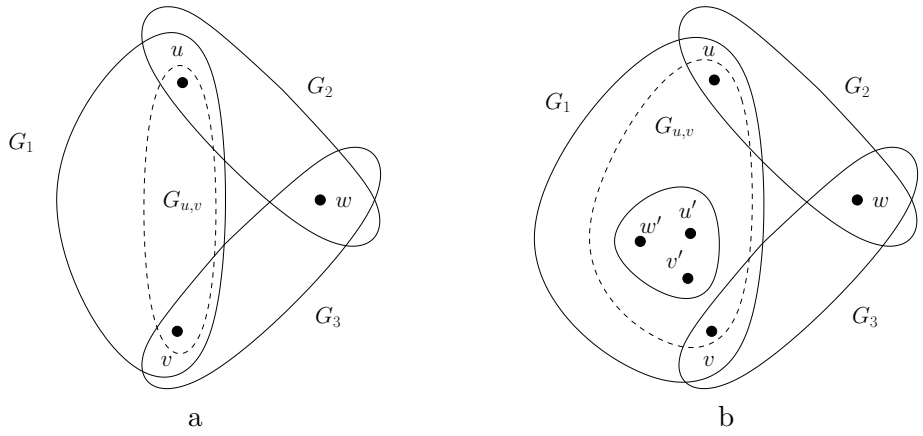


Figure 3.5: Illustration of Theorem 3.5.5. a) The minimum subgraph $G_{u,v}$ including u, v is included in the cluster G_1 which contains u and v. b) Tree-Decomposition Step (u, v, w) depends indirectly on every tree-decomposition step in $G_{u,v}$.

including u, v but not including the hinges u', v', w', see Figure 3.5b. This proves that T_i depends indirectly on the nodes in $\mathcal{H}_{u,v}(G)$.

Besides, the hinges of a tree-decomposition step T_j in G_1 but not in $G_{u,v}$ are included in G_1 but not in $G_{u,v}$. Then, there exists a tree-decomposable Laman subgraph, $G_{u,v}$, including u, v but not the hinges of T_j. The tree-decomposition step is then independent from T_i. □

Subgraphs $\mathcal{H}_{u,v}(G)$ spanned by a pair of vertices $u, v \in G$ are subgraphs of $\mathcal{H}(G) = (\mathcal{V}, E_D, E_S)$, and therefore can be spanned by the set of its representative nodes $R \subset \mathcal{V}$. It is easy to see that either $|R| = 1$ or $|R| = 2$. By definition, $\mathcal{H}_{u,v}(G)$ is the minimum complete subgraph of $\mathcal{H}(G)$ such that there exist nodes $V_1, V_2 \in \mathcal{V}$ with $u \in V_1, v \in V_2$. Then, the minimum subgraph spanned by V_1, V_2 is $\mathcal{H}_{u,v}(G)$.

We prove now that the representative nodes of the three h-graphs spanned by two of the three hinges of a tree-decomposition step T_i are the nodes in which T_i depends strongly.

Theorem 3.5.6
Let $G = (V, E)$ be a tree-decomposable Laman graph and $\mathcal{H}(G)$ the associated h-graph. Let T be a tree-decomposition of G, and T_i a tree-decomposition step in T with hinges u, v, w. Let $\mathcal{H}_{u,v}(G)$ be the graph spanned by the two vertices u, v. Then T_i strongly depends on the representative vertices of $\mathcal{H}_{u,v}(G)$.

Proof
Graph $\mathcal{H}_{u,v}(G) = (\mathcal{V}, E_D, E_S)$ is the minimum graph such that there exist nodes $V_1, V_2 \in \mathcal{V}$

with $u \in V_1, v \in V_2$. Consider R the set of representative nodes of $\mathcal{H}_{u,v}(G)$, then $|R| = 1$ or $|R| = 2$. Consider an element $R_0 \in R$, we prove that T strongly depends on R_0.

We show first that either R_0 contains u or v, or it exists a step T_0 wich contains u or v and R_0 depends indirectly on it. We show, by contradiction and for the two possible cases $|R| = 1$ and $|R| = 2$, that in the case that R_0 does not contain u nor v, a step T_0 like the one described exists.

If $|R| = 1$, R_0 does not contain u nor v, and no step T_0 in which R_0 depends indirectly contains u nor v, then it is impossible that R_0 spans the subgraph $\mathcal{H}_{u,v}(G)$, because it does not contain u nor v. This is a contradiction, and then this case is proven.

Consider $|R| = 2$, with $R = \{R_0, R_1\}$, R_0 does not contain u nor v, and no step T_0 in which R_0 depends indirectly contains u nor v. Consider the minimum path $P = \{P_i\}_{i=0}^{n}$ in $\mathcal{H}_{u,v}(G)$ between R_0 and R_1, such that $P_0 = R_0$ and $P_n = R_1$. Then, the graph spanned by nodes P_1, P_n will include all the elements in $\mathcal{H}_{u,v}(G)$ but the ones included in the graph spanned by P_0, $\mathcal{H}_{P_0}(G)$. In particular it will include elements u, v, because they are not included in $\mathcal{H}_{P_0}(G)$.

The graph spanned by nodes P_1, P_n is a subgraph including u, v and either is the same or is smaller than $\mathcal{H}_{u,v}(G)$. In the case that it is the same, P_0 can not be a representative node of the graph, which is a contradiction, because the distance from P_1 to P_n is less than the distance from P_0. In the case that it is smaller, since $\mathcal{H}_{u,v}(G)$ is the minimum graph by defintion, we find a new contradiction, and then this case is also proven.

We show now by contradiction that there is no tree-decomposition step T_j such that T_i depends indirectly on T_j and T_j depends indirectly on R_0, for the two possible cases $|R| = 1$ and $|R| = 2$. Consider then that there exists a tree-decomposition step T_j such that T_i depends indirectly on T_j and T_j depends indirectly on R_0.

If $|R| = 1$, $R_0 \in R$ spans $\mathcal{H}_{u,v}(G)$. Since T_j depends indirectly on R_0, T_j can not be included in the graph spanned by R_0, $\mathcal{H}_{u,v}(G)$. Then, by Theorem 3.5.5, T_j is independent on T_i, which is a contradiction. This case is not possible then.

If $|R| = 2$, and T_j depends indirectly on R_0, T_j must be in the minimum path between R_0 and R_1. Otherwise, the nodes in this minimum path together with the nodes in which they depend indirectly will span a graph, including R_0 and R_1, in which T_j is not included, which is a contradiction. Then the representative node of $\mathcal{H}_{u,v}(G)$ must be T_j instead of R_0, as every graph containing T_j will also contain R_0. This case is also a contradiction, which proves that no such a tree-decomposition step T_j may exist. □

We have shown that a tree-decomposition step merging three clusters by means of hinges u, v, w strongly depends on the representative nodes of the minimum graphs spanned by the three possible pairs of its hinges. We present in Algorithm 2 a method to compute

the strong dependences of a tree-decomposition step with respect to one cluster using Theorem 3.5.6. Given two hinges, the algorithm first finds the minimum path between two nodes in the h-graph each one including one of the two hinges. The found path must be included in the minimum subgraph M spanned by the two hinges, and must include the representative nodes of M. In order to find them, the algorithm checks if the graph spanned by the different elements in the path is M, and takes the two closer ones for which this holds.

Algorithm 2 Compute strong dependences

```
Input: HG, the h-graph of the cluster whose dependences we search for
       u, v, the hinges included in HG
Output: R1, R2 the strong dependences of the step

function Compute_Strong_Dependences()
U := Set of nodes in HG including u
V := Set of nodes in HG including v
P := Minimum path between any two nodes, one in U, one in V
while P.length ≥ 1 and ( GraphSpannedBy( P[1], P[P.length] ) == GraphSpannedBy(
P[0], P[P.length] )) do
  P.delete[0]
end while
while P.length ≥ 1 and ( GraphSpannedBy( P[0], P[P.length - 1] ) ==
GraphSpannedBy( P[0], P[P.length] )) do
  P.delete[P.length]
end while
return P[0], P[P.length]
endfunction
```

Consider the graph spanned only by one node, say v. This graph only includes the nodes which are essential for the construction of node v, which in practice corresponds to the constructions steps necessary to construct v. That is, the graph spanned by one node v only includes the nodes in which v depends indirectly, either strongly or not. This is a very interesting property which will be useful to find the nodes in which a given node depend. To compute this graph, we have developed a recursive algorithm which determines the set of clusters in which v depends indirectly. Then, for every node in a cluster, we apply the same algorithm until the node under study has no indirect dependences.

3.6 Conclusions

In this chapter we have introduced h-graphs, a new representation for tree-decomposable Laman graphs which summarizes in the same graph the information about the graph and its tree-decomposition. This later feature, and the way they explicit the relations of dependence between the decomposition steps of the graph, make h-graphs very efficient in order to search for certain sub-structures in tree-decomposable Laman graphs.

We have shown different ways to construct the h-graph associated to a tree-decomposable Laman graph. Basically, it can be constructed at the same time than the graph, when the graph is constructed from the triangle graph K_3 by means of a constructive sequence, or from the construction plan or tree-decomposition of the graph, once the graph has been already tree-decomposed. The main drawback of h-graphs is that, for a given graph G of any order, to build the h-graph is equivalent to decomposing G. Also, an analysis could be done to determine if some primitives of h-graphs could be speeded-up.

CHAPTER 4

Parameter ranges

Every word or concept, clear as it may seem to be,
has only a limited range of applicability.

Werner Heisenberg

The computation of parameter ranges in which the geometric constraint problems with one degree of freedom or variant parameter have an actual realization is a challenging and longtime addressed problem. In dynamic geometry, the computation of the domain, which is the set of parameter ranges for each one of the possible index assignments where the constraint problem solution is actually realizable, is an essential step in the process leading to the solution of the reachability problem.

Until the publication in 2008 of the van der Meiden works, [103, 105], the search for parameter ranges in which the geometric constraint problem have an actual realization was made by regular sampling. But in his PhD dissertation [103], van der Meiden presents a simple method which computes efficiently the parameter ranges of a given geometric constraint problem. Despite the rigorous description of the approach, no correctness proof is presented in [103] and no implementation evidence is given.

In this chapter we will study the van der Meiden method, presenting an accurate correctness proof, the key points of our implementation and a case study which will illustrate

the whole process. The result of this work can also be found in Hidalgo *et al.* [37, 42].

4.1 Preliminaries

The van der Meiden method is an algorithm which efficiently computes the parameter ranges of a given geometric constraint problem. It is based on the decomposition of the problem in different tree-decomposition steps and in identifying the dependence of these tree-decomposition steps with respect to the variant parameter.

In this section we present some concepts necessary for the complet understanding of the van der Meiden method, as well as a formal definition of the domain of a geometric constraint problem with one degree of freedom.

4.1.1 The domain of a geometric constraint problem with a variant parameter

In geometric constraint problems with one variant parameter, the construction feasibility of the problem depends on the value of the variant parameter. As defined in Section 2.3.1, Chapter 2, critical values of the variant parameter are those values in which the feasibility of the problem changes.

The domain of the geometric constraint problem with one variant parameter is defined as the set of values for which a construction plan for the problem is feasible. In general, the domain of a geometric problem is made of a set of disjoint intervals, each bounded by critical variant parameter values. We formalize now these concepts following to Freixas *et al* [25].

Definition 4.1.1
Let $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ *be a geometric constraint problem with one variant parameter* λ *and let* $T(\sigma, \lambda)$ *be a construction plan that solves* Π. *Given an index assignment* σ_j, *we define a* domain interval *of the variant parameter* λ *as the i-th connected set* $D^i_j \subseteq \mathbb{R}$, *such that for all* $\lambda \in D^i_j$ *the construction plan* $T(\sigma_j, \lambda)$ *is feasible.*

Notice that a domain interval D^i_j bounded by the critical values λ_l and λ_u is closed in λ_l or in λ_u if $T(\sigma_j, \lambda_l)$ or $T(\sigma_j, \lambda_u)$ are instances of the construction plan $T(\sigma, \lambda)$, respectively. Otherwise, it is open at that points.

Definition 4.1.2
Let $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ *be a geometric constraint problem with one variant parameter*

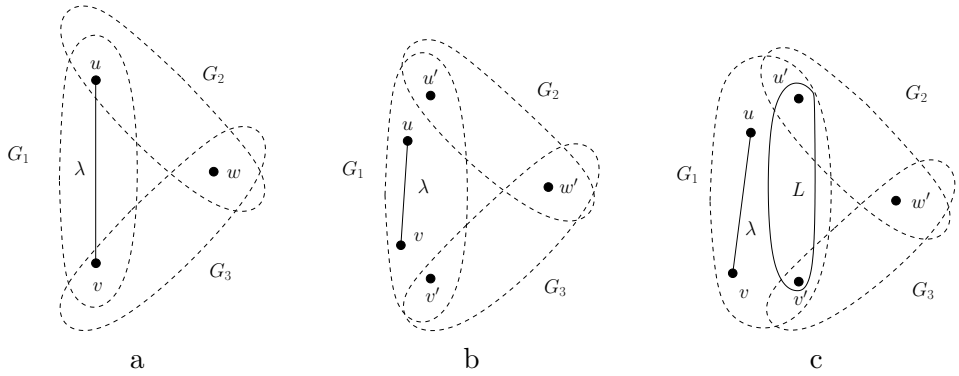


Figure 4.1: Dependency of a construction step. a) Directly dependent. b) Indirectly dependent. c) Independent.

λ *and let* $T(\sigma, \lambda)$ *be a construction plan that solves* Π. *We define the* domain *of* λ *as the union of domain intervals for all possible index assignments in the construction plan,* $D(\lambda) = \cup_{i,j} D^i_j(\lambda)$.

This definition of domain fits perfectly within the abstract notion given in Section 2.3.1, Chapter 2.

4.1.2 Dependence on the variant parameter

From now on, we consider that the geometric constraint problem $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ is represented by the graph $G = (V, E)$, where $V = \Pi_G$ are the geometric elements in Π and $E = \Pi_C$ are the constraints defined among them. We assume that the variant parameter is an edge $\lambda = (u, v) \in E$ and a generation sequence for G has been computed.

Dependency of a tree-decomposition step on the variant parameter is a central concept in this work. Each tree-decomposition step in a tree-decomposition will depend on the variant parameter in one of the following three different ways illustrated in Figure 4.1.

We define these concepts in the following definitions. The first definition states the concept of direct dependence of a tree-decomposition step with respect to the variant parameter.

Definition 4.1.3
Let $G = (V, E)$ *be a tree-decomposable Laman graph representing a geometric constraint problem with variant parameter* $\lambda = (u, v) \in E$. *Consider a tree-decomposition step* T_i *of a tree-decomposition* T *with hinge triple* $\{u, v, w\}$. *Then, we say that* T_i *depends directly on the variant parameter* λ.

Direct dependency is the simplest one of the two types of dependency. It is illustrated in Figure 4.1a. Notice that every tree-decomposition step including hinges u, v depends directly on the variant parameter $\lambda = (u, v)$.

Computing critical values of the variant parameter in directly dependent tree-decomposition steps is straightforward since a feasibility rule can be defined for each tree-decomposition step, depending on the constraints defined between the different elements. One can construct a dictionary, see Table 2.1, with the different critical values of the variant parameter for each tree-decomposition step, as seen in Section 2.3.1. The dictionary will be used subsequently in Section 4.2 in the van der Meiden method.

Definition 4.1.4
Let $G = (V, E)$ be a tree-decomposable Laman graph representing a geometric constraint problem with variant parameter $\lambda = (u, v) \in E$. Consider the tree-decomposition step T_i of a tree-decomposition T with hinge triple $\{u', v', w'\}$. Let G_1 be a cluster of the tree-decomposition step T_i. We say that the tree-decomposition step T_i depends indirectly on the variant parameter λ if $u, v \in G_1$ and there exists no subgraph $L \in \mathcal{T} \cup \mathcal{E}$ in G_1 such that $V(L)$ contains u', v' but not u, v.

Figure 4.1b illustrates this case. Indirectly dependent tree-decomposition steps represent the most important obstacle for a straightforward computation of the parameter ranges of a problem. In Section 4.2 we will explain how to solve this problem.

Finallywe define the problems which are independent on the variant parameter.

Definition 4.1.5
Let $G = (V, E)$ be a tree-decomposable Laman graph representing a geometric constraint problem with variant parameter $\lambda = (u, v) \in E$. Consider a tree-decomposition step T_i. We say that T_i is independent from the variant parameter λ if T_i does not depend neither directly nor indirectly on the variant paramter.

This situation is shown in Figure 4.1c. Notice that in independent tree-decomposition steps the construction can be actually carried out without taking into account the value of the variant parameter. Effectively, the placement of elements u', v' is fixed whenever the subgraph L is fixed. Then, the other two clusters not including L, clusters G_2 and G_3 in Figure 4.1a, can be merged with L regardless of the variant parameter's value.

4.1.3 Dependence and h-graphs

Dependence of the different tree-decomposition steps of a graph representing a geometric constraint problem with respect to the variant parameter of the problem can be determined efficiently using h-graphs, introduced in Chapter 3.

Consider the tree-decomposable Laman graph $G = (V, E)$ representing the geometric constraint problem $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ with variant parameter $\lambda = (u, v) \in E$. Consider now the h-graph $\mathcal{H}(G) = (\mathcal{V}, E_D, E_S)$ associated to G, as defined in Section 3.2. $\mathcal{V}$ is the set of hinge triples of G, representing the set of generation steps in any tree-decomposition T for G. E_D, E_S represent the different hierarchical dependences stablished among the generations steps.

Although dependences between two tree-decomposition steps, as defined in Section 3.1, Chapter 3, and dependences of a tree-decomposition step on the variant parameter represent different concepts and are used for different purposes, there is a clear analogy between them. In fact, we can determine the dependence of a tree-decomposition step on the variant parameter from the dependence between two tree-decomposition steps. We show that for each kind of dependence.

For direct dependences, assume that the variant parameter of the problem is the edge $\lambda \in E$. Consider two tree-decomposition steps of G which depend directly on each other. By Definition 3.1.1, both tree-decomposition steps share two hinges a, b, and the edge $(a, b) \in E$. If the edge (a, b) is the variant parameter λ, then both tree-decomposition steps depend directly on the variant parameter λ.

For indirect dependences, let $V_1 \in \mathcal{V}$ be a tree-decomposition step, represented by the hinge triple (u, v, w), which depends directly on the variant parameter $\lambda = (u, v)$. Let $V_2 \in \mathcal{V}$ be a tree-decomposition step which depends indirectly on V_1. By Definition 3.1.2, one of the clusters of V_2, say G_1, includes the three hinges of V_1, in particular u and v, and the edge defined by them. Besides, no Laman subgraph L exists like the one in Definition 4.1.5. Then, the variant parameter $\lambda = (u, v)$ is included in G_1, and thus, V_2 depends indirectly on the variant parameter λ.

For independent problems, let $V_1 \in \mathcal{V}$ be now a tree-decomposition step, represented by the hinge triple $\{u, v, w\}$, which depends directly on the variant parameter $\lambda = (u, v)$. Let $V_2 \in \mathcal{V}$ be a tree-decomposition step represented by the hinge triple $\{u', v', w'\}$ which does not depend on V_1. By Definition 3.1.4, one of the clusters of V_2, say G_1, includes the three hinges of V_1, the edges defined among them, and a Laman subgraph L including two hinges, say u', v', but not u, v. Then, G_1 includes the variant parameter $\lambda = (u, v)$ and a Laman subgraph L like the one in Definition 4.1.5, and thus, V_2 depends indirectly on the variant parameter λ.

Since h-graphs are precisely graphs which represent the different kinds of dependences arising between the tree-decomposition steps of a tree-decomposable Laman graph, they can be used to determine the kind of dependence of the tree-decomposition steps with respect to the variant parameter. The dependence of any tree-decomposition step with respect to the variant parameter is computed following a three steps algorithm.

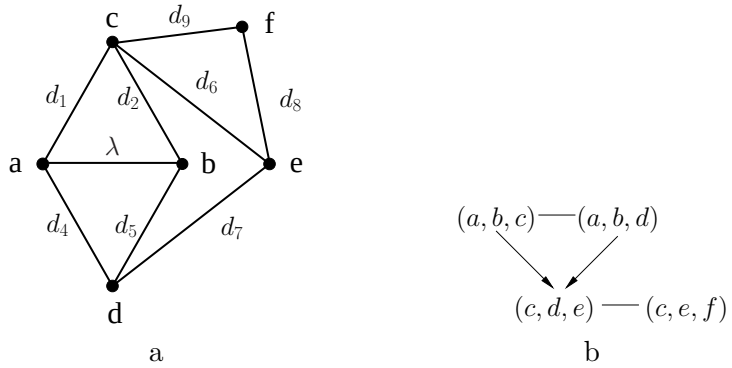


Figure 4.2: Dependence on the variant parameter. a) Graph G representing the geometric constraint problem Π_λ. b) H-graph $\mathcal{H}(G)$ associated to G.

First, if the variant parameter is defined upon vertices u, v, we can use the h-graph to find the set of nodes including u, v as hinges. These nodes are those depending directly on the variant parameter. We call the set of such nodes $\mathcal{D}$.

Then, for every node V in $\mathcal{D}$, we use the h-graph to find all the nodes V_R depending indirectly on V. Every node V_R joined with V by means of a concatenation of s-edges depends indirectly on V. Nodes V_R will depend indirectly on the variant parameter.

Finally, all nodes not included in $\mathcal{D}$ nor depending indirectly on any node in $\mathcal{D}$ are independent on the variant parameter.

Algorithm 3 shows the algorithm used to compute the dependences of the tree-decomposition steps of a tree-decomposition of a graph G by means of the h-graph $\mathcal{H}(G)$. HG represents a h-graph and stores the sets HV, ED and ES standing for the nodes set, the set of direct dependences and the set of strong dependences respectively. The algorithm returns two sets, D and S, including the nodes depending directly and indirectly on the variant parameter, respectively. The nodes in HV not included in D or I are independents on the variant parameter.

To illustrate how the algorithm works, consider the graph G depicted in Figure 4.2a representing a geometric constraint problem with variant parameter $\lambda = (a, b)$, and the associated h-graph $\mathcal{H}(G)$ in Figure 4.2b. Nodes in $\mathcal{H}(G)$ including hinges a, b depend directly on λ. These nodes are $(a, b, c), (a, b, d)$. Nodes depending indirectly on nodes $(a, b, c), (a, b, d)$ depend indirectly on λ. There is just one such node, (c, d, e). Finally, nodes independent from nodes $(a, b, c), (a, b, d)$ are also independent on λ. Node (c, e, f) is then independent on the variant parameter λ.

Algorithm 3 Compute the type of dependency

```
Input: HG = ( HV, ES, ED ), the h-graph associated to G
       u, v, the vertices upon which stands the variant parameter
Output: ( DD, ID ) sets including the directly and indirectly dependent nodes

function Compute_Dependence
DD = Elements in HV incuding u, v
ID = ∅
S = DD
while Size of S > 0 do
  E = Edges in ES beginning at S[ 0 ]
  for each E[ j ] in E do
    S = S ∪ { Opposite to S[ 0 ] in E[ j ] }
    ID = ID ∪ { Opposite to S[ 0 ] in E[ j ] }
  end for
  Remove S[ 0 ] in S
end while
return ( DD, ID )
```

4.2 The van der Meiden method

The van der Meiden method to compute the domain of the variant parameter is based on the identification of a set of points which *could* be the endpoints of the parameter domain intervals. These points will be referred to as *candidate points*.

The maximum possible domain of a problem, understood as the domain of a problem if no constraint restricts it, depends on the type of the variant parameter. Specifically, for distance type variant parameters the maximum possible domain is the set $\mathbb{R}^+$, as we only consider positive distances. For angle type variant parameters the maximum possible domain is the set $[-\pi/2, \pi/2]$, as we consider angles defined inside this interval. The computed candidate points will however split this maximum domain in different regions or intervals, where feasibility remains constant. This observation will be proven in Section 4.4. In a subsequent phase, the actual endpoints of the domain intervals are selected from the set of candidate points, determining the final domain of the problem.

The method has therefore two steps, [105]. In the first one the candidate points are computed. In this step index assignments are irrelevant. In the second step, we need to fix a specific index assignment and select from the set of candidate points which ones will determine the endpoints of the domain intervals associated to this concrete index assignment. The process must be repeated for every possible index assignment.

In this section we explain in detail the van der Meiden method. Firstly we introduce the method to determine the candidate points and then we explain the selection of the actual endpoints of the domain intervals. Finally we discuss the main drawbacks of the method.

4.2.1 Computing the candidate points

Since only one variant parameter is considered, degenerate situations appear only in those tree-decomposition steps that depend on it, whether directly or indirectly. Depending on the kind of dependency, two different procedures to determine the candidate points are considered.

For tree-decomposition steps (u, v, w) depending directly on the variant parameter $\lambda = (u, v)$, their critical values define the candidate points. In the case that the tree-decomposition step is a basic step, all edges between hinges exist and the critical values can be computed applying the dictionary that collects for each degenerate case a specific solution method. This dictionary is shown in Table 2.1, Section 2.3.1. These critical values are the candidate points generated by the step (u, v, w).

In the case that the tree-decomposition step is not a basic step, then at least one of the two edges $(u, w), (v, w)$ is missing. For each missing edge there is a tree-decomposable Laman subgraph including either u, w or v, w. Specifically, the distance or angle between the included pair of hinges will be measurable. Then, a basic equivalent tree-decomposition step can be constructed by replacing the cluster by the rigid bar (u, w) or (v, w) and assigning to this new constraint the measured value. The critical values of this new problem can be computed applying the dictionary shown in Table 2.1, Section 2.3.1, as above. These critical values are the candidate points generated by the step (u, v, w).

For indirectly dependent tree-decomposition steps, the computation of the critical values must be done indirectly. Consider a tree-decomposition step (u', v', w') which depends indirectly on the variant parameter $\lambda = (u, v)$. Assume that G_1 is the cluster merged by (u', v', w') including the variant parameter $\lambda = (u, v)$ and that the hinges included in G_1 are u', v'. The method to compute the critical values transforms in the first place the indirect dependence into a direct one. Constraint $\lambda = (u, v)$ is removed and a new variant parameter $\mu = (u', v')$ is added. In that way, direct dependence of (u', v', w') on the variant parameter is assured.

Then, critical values for this new modified problem are calculated as in the basic tree-decomposition step case by using the dictionary of direct dependency as described above. Finally, the modified problem is solved and constructed for each of those critical values, and the candidate points are computed by measuring the value of the relation between geometric elements u, v at each construction. Figure 4.3 shows the different steps of the

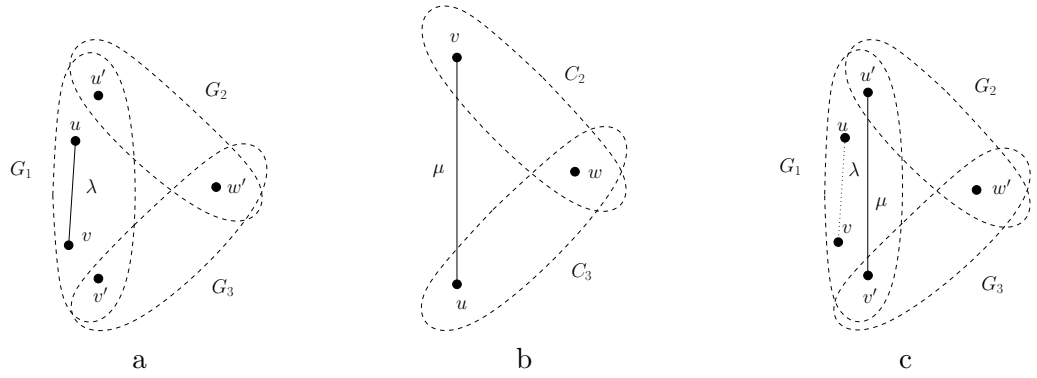


Figure 4.3: Candidate points computation process for indirectly dependent tree-decomposition steps. a) Tree-decomposition step that depends indirectly on λ. b) Transformed problem that depends directly on μ. c) Construction where values for the variant parameter λ are measured.

process.

Figure 4.4 shows the graphic of the relation between the values of the variant parameter λ and the values of μ. Notice that both variant parameters have different maxima and minima. Notice also that, as μ increases its value, λ increases rapidly up to a local maximum and then decreases slowly again. Finally, notice that the value of λ for the minimum and maximum values of μ is the same, zero.

Intuitively, the candidate points of an indirectly dependent tree-decomposition step are the values of λ measured when the new variant parameter μ takes critical values, that is, the values of λ for which the feasibility of the construction may change.

4.2.2 Computing the domain

Once all candidates have been computed, it is necessary to elucidate which of them are actually bounds of interval domains. At this point of the method a specific index assignment must be fixed, as computations to determine the domain include the actual construction of the problem.

The second step of the method consists on checking for the constructibility of the problem at each region resulting from the splitting of the maximum possible domain of the problem by the candidate points. We consider each interval defined by two subsequent critical values, pick in it a value for the variant parameter and check whether the construction plan is feasible or not.

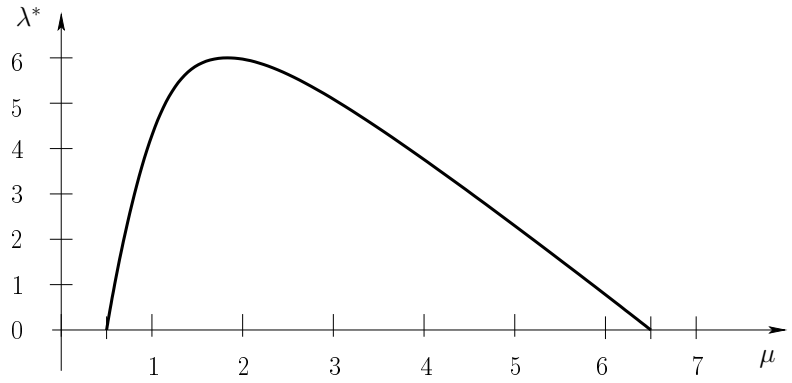


Figure 4.4: Relation between the values of the variant parameter λ and the values of μ.

The approach relies on the fact that, under the assumption that the variant parameter continuously changes within the interval bounded by two subsequent critical values, see Section 4.4, construction plan feasibility does not change inside such intervals. Therefore, checking for feasibility in one point within each such interval is enough to establish feasibility over it.

Besides, feasibility on the candidate points must also be checked to elucidate the topology of the feasible intervals or the existence of isolated points of feasibility. A feasible interval is closed at a bound whenever the construction at this bound is feasible. On the contrary, an interval is open at a bound if the construction at this bound is not feasible. Many configurations can appear, such as isolated points of feasibility or isolated points of unfeasibility. In principle, intuition suggests to consider as a single interval two feasible intervals sharing a feasible bound. A discussion on this topic is developed at the end of Section 5.1.3.

All this feasibility computations must have been done after the selection of a specific index assignment σ. The resulting domain intervals are the domain intervals associated to σ. For the computation of the domain, it is necessary to settle one by one all the possible index assignments of the problem and carry out the computation of their feasible intervals. The union of all feasible intervals for all possible index assignments will lead to the complete domain of the problem.

4.2.3 Limitations of the method

Analyzing the method proposed in this section, we easily see that, in order to compute the domain of the problem Π, the solver must solve not only Π but a number of new problems

derived from the indirectly dependent tree-decomposition steps defined in it. Indeed, for the method to succeed, all considered problems must be tree-decomposable.

Unfortunately, in the van der Meiden method there is no election in the change of driving parameter, and the definition of the different problems is given by the situation of the variant parameter with respect to the indirectly dependent tree-decomposition steps. The tree-decomposability can not be then assured. Moreover, chances are that as the problem size increases, so does the possibility of including transformed problems which are not tree-decomposable. In the case that one of the derived problems is not tree-decomposable, either a different solving approach is applied to solve the transformed problem or the method fails.

4.3 Our implementation

Let T be the decomposition tree that solves the constraint problem at hand. In our implementation, each node in T stores:

1. T.*built*: A boolean flag that takes value true whenever the coordinates of the geometric object have been actually computed with respect to a local framework,
2. T.*coordinates*: An array of coordinates that places every geometric object with respect to a local framework,
3. T.*rule*: An identifier for the solving rule. If the node is a leaf, the rule is a basic placement. Otherwise the rule identifies the merging of three clusters,
4. T.*hinges.u*, T.*hinges.v*, T.*hinges.w*: Pointers to the hinges in the set of geometric objects on which the merging was carried out, and
5. T.$left$ and T.*right* point to the left and right tree child respectively.

Without loss of generality and for the sake of simplicity and convenience, we assume that the variant parameter is always defined upon hinges u and v in the leftmost cluster in each set of sibling nodes of a tree-decomposition step of T. Since we consider problems with just one variant parameter and the order in which siblings are depicted in the decomposition tree is meaningless, this assumption just implies that the tree has been conveniently rewritten.

We assume that the set of geometric elements, G, the set of constraints, C, the vector of parameters in C, P, the index in P of the variant parameter, i, and, the decomposition tree of the problem being solved, T are stored as static variables. Moreover, T.*built* value is true for the leaf nodes and false otherwise. Our implementation is based on the algorithms listed in Algorithm 4 through Algorithm 6.

Algorithm 4 Computing de Domain

```
Input: HG = ( HV, ES, ED ), the h-graph associated to G
       T, the tree-decomposition of G
       ( u, v ), the variant parameter of G
Output: The domain of the problem represented by G

function Compute_Domain()
D := ∅
( DD, ID ) = Compute_Dependence( HG, u, v )
K := Compute_Critical_Values( T, DD, ID )
for each possible index assignment σ do
  for each subsequent interval [ c1, c2 ] in K do
    p = ( c1 + c2 )/ 2
    if T( p, σ ) is feasible then
      Check for feasibility at interval bounds c1 and c2
      I = Interval from c1 to c2, with the measured topology
      D = D + I
    end if
  end for
end for
return D
endfunction
```

Algorithm 5 Computing Critical Values

Input: The subtree T whose root is the current tree-decomposition step
DD, the set of nodes directly depending on the variant parameter
ID, the set of nodes indirectly depending on the variant parameter
λ = (u, v), the variant parameter of G
Output: The set of critical values K

function Compute_Critical_Values()
if not T.left.built **then**
Compute_Critical_Values(T.left)
end if
if T.left in DD **then**
K := K + Critical_Values_From_Dictionary(T.rule, (u, v))
else if T.left in ID **then**
μ = Dummy_Parameter(T.hinges.u, T.hinges.v)
P_μ = { P' - λ } $\cup$ { μ }
T_μ = Solve_Problem(G, C, P_μ)
σ := Significative index associated with the dummy parameter μ
Q := Critical_Values_From_Dictionary(T_μ.rule, μ)
for each q in Q **do**
for each possible index assignment σ_i in σ **do**
R = Build_Realization(T_μ, q, σ_i)
K = K + Measure_In_Realization(R, λ)
end for
end for
end if
T.built := **true**
return K

Algorithm 6 Compute Dummy Parameters

Input: $u, v,$, hinges in cluster C_1
Output: Type of needed dummy constraint

function Dummy_Parameter
if IsApoint(u) **and** isApoint(v) **then**
return point-point-distance
else if (IsApoint(u) **and** isAline(v)) **or** (IsApoint(v) **and** isAline(u)) **then**
return point-line-distance
else if IsAline(u) **and** isAline(v) **then**
return line-line-angle
end if

4.4 Algorithm correctness

In this section we show the correctness of our algorithm. In [29], a general result concerning the computation of critical values of constraint problems over points and distances constraints with one degree of freedom is presented. Here we consider problems which include distance and angle constraints and the underlying graphs must be Laman and tree decomposable, that is, a super set of Henneberg-I graphs. The results presented in Sections 4.4.1 and 4.4.2 are collected in [41].

4.4.1 The transformation

We shall start the van der Meiden method correctness proof by showing that the transformation defined in Section 4.2 is always possible. That is, the constraint corresponding to the variant parameter in the problem can always be replaced with another constraint. We will see that, in an indirectly dependent tree-decomposition step, no constraint is defined between the two hinges upon which the new variant parameter will be defined. Then, when the van der Meiden method is applied, the new variant parameter can be defined because it did not exist before.

Theorem 4.4.1
Let $G = (V, E)$ be a tree-decomposable Laman graph associated to the geometric constraint problem Π with one variant parameter $\lambda \in E$. Let (u, v, w) be a generation step of G which depends indirectly on the variant parameter and G_1 the cluster in which λ is included. If u, v are the hinges included in G_1 then no constraint is defined between hinges u and v.

Proof
For a contradiction assume that there is a constraint defined between hinges u and v, say $e = (u, v) \in E$. Then $e \neq \lambda$, otherwise the construction step would depend directly on e. Define the cluster $L \subset G_1$ including only edge e and vertices u, v. Then, L is an element in $\mathcal{E}$ and therefore (u, v, w) does not depend on λ. We have found the contradiction. □

Consider a generation step (u', v', w') of the graph G which depends indirectly on the variant parameter λ. Then, Figure 4.5 shows the only two possible locations for λ inside the problem. Since no constraint is defined between the two hinges in the cluster in which λ is included, u' and v', λ in particular can not be defined upon these two hinges.

This result assures the feasibility of the variant parameter change in the case of an indirect dependency. Since there exists no constraint defined upon the desired hinges, the removal of the original variant parameter and the addition of the new one is always possible.

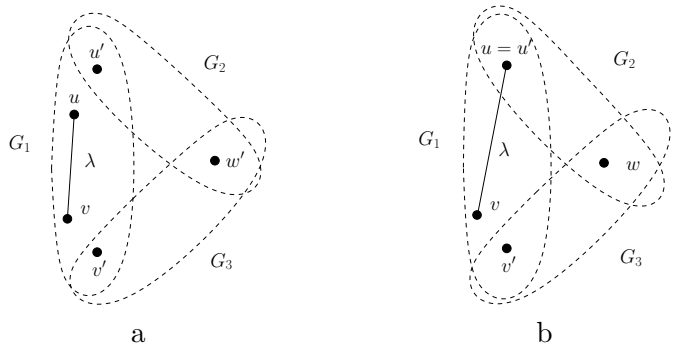


Figure 4.5: In well-constrained problems which indirectly depend on the variant parameter λ, the only two possible locations for λ are shown in this Figure. a) The variant parameter is defined upon two elements different to the hinges u', v'. b) The variant parameter is defined upon one of the hinges, say u', and another arbitrary element different to the other hinge v'.

4.4.2 The set of solution instances

As seen in Section 4.2, in case that a problem depends indirectly on the variant parameter, the van der Meiden method computes ranges for feasible values of variant parameters transforming the problem by removing the variant parameter in the given problem and adding a convenient, new variant parameter. Thus, we need to prove that the sets of solution instances for the given problem and for the transformed one are the same set.

Let $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ be a well-constrained geometric constraint problem and T a construction plan that solves Π. We shall denote by $\mathrm{T}(P, I)$ the instance of T resulting from evaluating T for the specific values in Π_P of the parameters in the constraints Π_C and index signs fixed in I. We will denote by $\mathcal{T}$ the set of all possible instances $\mathrm{T}(P, I)$ for problem Π. We start the proof by stating a trivial lemma.

Lemma 4.4.2
Let $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ *be a well-constrained geometric constraint problem. Let* T *be a construction plan that solves* Π *and let* $\mathrm{T}(P, I)$ *be a realization of the solution instance. Then for any pair of geometric objects* $g_i, g_j \in G$ *any relationship between them can be measured in* $\mathrm{T}(P, I)$.

Proof
Since the problem is well-constrained, any realization $\mathrm{T}(P, I)$ places all the geometric elements in G with respect to a common reference. □

Next we state and prove a lemma that relates solution instances for different geometric

constraint problems defined on the same set of geometric elements for which tree decompositions are known.

Lemma 4.4.3
Let $\Pi_1 =< \Pi_G, \Pi_C^1, \Pi_P^1 >$ *and* $\Pi_2 =< \Pi_G, \Pi_C^2, \Pi_P^2 >$ *be two well-constrained geometric constraint problems defined on the same set of geometric elements* G*. Let* T_1 *and* T_2 *be construction plans that respectively solve problems* Π_1 *and* Π_2 *and let* $\mathrm{T}_1(P_1, I_1)$ *be a solution instance for the problem* Π_1*. If parameters in* Π_P^2 *and indices in* I_2 *are assigned values measured in the actual solution* $\mathrm{T}_1(P_1, I_1)$*, then* $\mathrm{T}_2(P_2, I_2)$ *is a solution instance for* Π_2*.*

Proof
By Lemma 4.4.2, we can measure a value for any constraint parameter in C_2 and derive any sign in I_2 in the actual solution instance $\mathrm{T}_1(P_1, I_1)$. The construction plan $\mathrm{T}_2(P_2, I_2)$ represents all possible solution instances to problem Π_2 and in particular the solution instance corresponding to values of P_2 and I_2 measured from the actual construction $\mathrm{T}_1(P_1, I_1)$. □

To illustrate this lemma, consider a set of geometric objects including four points p_1, p_2, p_3 and p_4 and two lines l_1, l_2. Now consider two different geometric constraint problems defined on them, say Π_1 and Π_2. Problem Π_1 the constraint graph of which is depicted in Figure 4.6a includes six point-point distance constraints, $d_1^1, d_2^1, d_3^1, d_4^1$ an angle constraint a_1^1 plus four point-on-line incidence. If constraints are given values $d_1^1 = 5, d_2^1 = 5, d_3^1 = 8, d_4^1 = 4$ and $a_1^1 = 53.13$, Figure 4.7a shows an instance solution to problem Π_1.

Problem Π_2 whose graph depicted in Figure 4.6b is defined by means of five point-point distances $d_1^2, d_2^2, d_3^2, d_4^2, d_5^2$ plus four point-on-line incidences. By Lemma 4.4.2 we can measure any relationship between the geometric elements placed with respect each other in the solution instance of Π_1 shown in Figure 4.7a. If constraints in problem Π_2 are assigned values measured in this way, we get $d_1^2 = 4, d_2^2 = 4, d_3^2 = 8, d_4^2 = 8.94, d_5^2 = 6.4$. In these conditions problem Π_2 is feasible. If additionally we measure in the Π_1 solution instance signs corresponding to the step constructions in Π_2 with more than one possible solution, we get the specific solution instance to Π_2 show in Figure 4.7b.

Finally, we prove that the set of solution instances of both the original problem and the modified one are the same.

Theorem 4.4.4
Let $\Pi_\lambda =< \Pi_G, \Pi_C^\lambda, \Pi_P^\lambda >$ *and* $\Pi_\mu =< \Pi_G, \Pi_C^\mu, \Pi_P^\mu >$ *be two well-constrained geometric constraint problems, defined on the same set of geometric elements* Π_G *with variant parameters* λ *and* μ *respectively, and such that* $\Pi_C^\lambda - \{\lambda\} = \Pi_C^\mu - \{\mu\}$*. Moreover, let* T_λ *and* T_μ *be construction plans that respectively solve* Π_λ *and* Π_μ*. Then the sets of solution instances* $\mathcal{T}_\lambda$ *and* $\mathcal{T}_\mu$ *for problems* Π_λ *and* Π_μ *generated by arbitrarily varying* λ *and* μ *respectively*

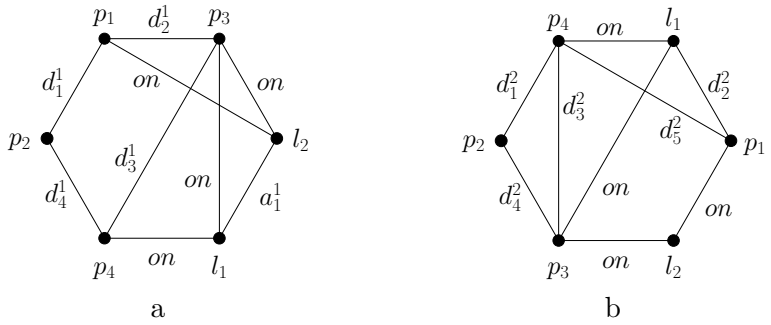


Figure 4.6: Two problems defined over the same set of geometric objects. a) Problem Π_1. b) Problem Π_2.

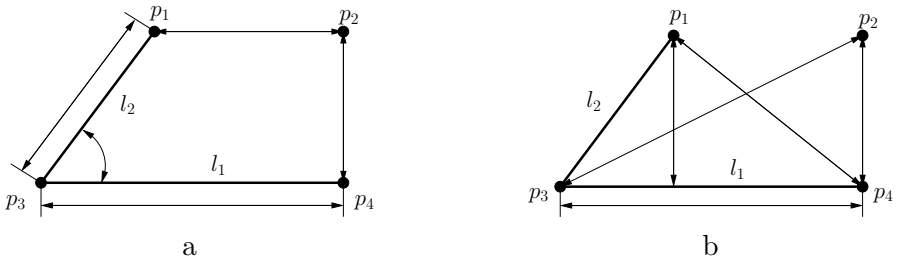


Figure 4.7: Solution instances. a) Instance for problem Π_1. b) Instance for problem Π_2.

are the same set.

Proof
We will prove $\mathcal{T}_\lambda = \mathcal{T}_\mu$ by a double inclusion. First we take an arbitrary element on $\mathcal{T}_\lambda$ and show that it is also contained in $\mathcal{T}_\mu$. Consider the instance $T_\lambda(P_\lambda, I_\lambda) \in \mathcal{T}_\lambda$. It fulfills all common constraints in Π_λ and Π_μ, which are the fixed constraints in problem Π_μ. By Lemma 4.4.2, we can measure the value of the constraint corresponding to μ. Since $T_\lambda(P_\lambda, I_\lambda)$ actually exists, this value will be a real number and, by Lemma 4.4.3, the instance will be also instance of problem Π_μ. Then $T_\lambda(P_\lambda, I_\lambda) \in \mathcal{T}_\mu$ and the inclusion is proved. The reverse inclusion is made analogously, proving that $\mathcal{T}_\mu \subseteq \mathcal{T}_\lambda$. That completes the proof. □

This result states that two problems defined on the same set of geometric elements whose constraints only differ in the variant parameter have always the same set of possible instances. In other words, this assures that when a problem has one degree of freedom, the constraint considered as variant parameter does not matter. The set of solution instances generated by the variation of the value of the degree of freedom in the resulting construction is always the same.

4.4.3 Correctness

In this last section we finally prove that the algorithm proposed by van der Meiden in [103] is correct and it finds the parameter ranges of a geometric constraint problem. First we will see that values of the original variant parameter measured at critical values of the transformed problem are critical values of the given problem.

Lemma 4.4.5
Let $\Pi_\lambda =< \Pi_G, \Pi_C^\lambda, \Pi_P^\lambda >$ *be a geometric constraint problem and* P_i *a tree-decomposition step that indirectly depends on* λ. *Let* $\Pi_\mu =< \Pi_G, \Pi_C^\mu, \Pi_P^\mu >$ *be the problem resulting after replacing* λ *by* μ, *in such a way that* P_i *directly depends on* μ. *Let* T_μ *be a solution tree to* Π_μ *for some index* I_μ, *and let* μ^* *be a critical value of* T_μ. *If* $T_\mu(\mu^*)$ *exists and* λ^* *is the value of parameter* λ *measured in it, then constructibility of* T_λ *changes at* λ^*. *That is,* λ^* *is a critical value for the problem* Π_λ.

Proof
Let μ^* be a critical value for Π_μ and λ^* be the measure in T_μ for parameter λ. Assume that C_1, C_2 and C_3 are the three clusters involved in the indirectly dependent problem with C_1 being the cluster that undergoes the problem transformation.

Since the problem is well-constrained, and according to Theorem 4.4.4, continuously changing λ in cluster C_1 of T_λ, will result in λ reaching the value λ^*. Then, μ will take the value μ^* and constructibility of T_λ will change accordingly to the constructibility of the

problem defined over C_1, C_2 and C_3. Therefore λ^* is a critical value for Π_λ. □

This result assures that every critical value in the modified problem is also a critical value in the original one, as long as the construction is feasible at that point. With all these preliminary results, we will show now that the proposed algorithm figures out all the critical values and only critical values of a geometric constraint problem.

Theorem 4.4.6
Let $\Pi_\lambda =< \Pi_G, \Pi_C^\lambda, \Pi_P^\lambda >$ *be a geometric constraint problem and* T_λ *be a construction plan that solves* Π_λ*. Then, Algorithm 4 computes exactly the set of critical values of* Π_λ*.*

Proof
Correctness: We show that every point returned by the system is a bound in the domain of the problem. This is assured by the second part of the algorithm, which tests feasibility at the critical values bounding an interval domain and in a point within the interval. Each critical value λ_c separates two intervals: the one ending in it, say D_1, and the one beginning in it, say D_2. If a λ_c is not a bound of an interval of the domain of the problem, feasibility at D_1 and at D_2 will be the same, and the system will not pick it up as bound. Then the algorithm only picks up correct points.

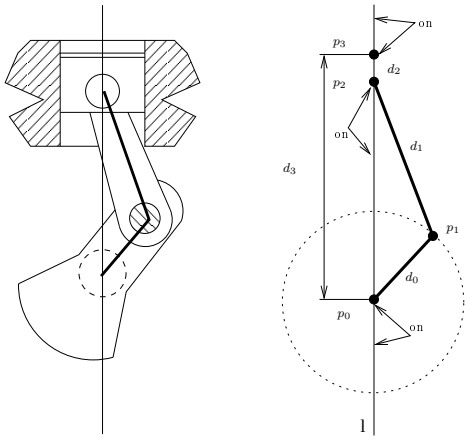
Completeness: We show now that every bound of any interval domain of the problem is selected. Effectively, a bound of an interval domain the domain of the problem must be critical value of at least one of its tree-decomposition steps. Since the algorithm returns all critical values of each considered tree-decomposition step, and it visits once and only once each tree-decomposition step in T_λ that is directly or indirectly dependent on λ, it returns necessarily all bounds of the problem's domain. □

This theorem proves that the van der Meiden method given in [103] effectively is correct and computes the feasible ranges of problems with one variant parameter.

4.5 Case study

To further illustrate how our algorithm works, we develop the piston and connecting rod crankshaft problem as a case study. This is a good example to highlight the full methodology of our algorithm as it involves all kinds of dependence in its tree-decomposition steps.

The case study we develop is depicted in Figure 4.8. From left to right, the figure shows a piston and connecting rod crankshaft, an abstraction of the piston represented as a geometric constraint problem, and an actual construction plan that solves the problem. The set of geometric elements includes four points and a straight line $\Pi_G = \{p_0, p_1, p_2, p_3, l\}$. The set of constraints includes four point-point distances and three



a b

1. $p_0 = origin()$
2. $p_3 = distD(p_0, d_3)$
3. $l = line2P(p_0, p_3)$
4. $c_0 = circleCR(p_0, d_2)$
5. $p_2 = iLC(l, c_0, s_1)$
6. $c_1 = circleCR(p_0, d_0)$
7. $c_2 = circleCR(p_2, d_1)$
8. $p_1 = iCC(c_1, c_2, s_2)$

c

Figure 4.8: Case study. a) Piston and connecting rod crankshaft. b) Geometric abstraction. c) Construction plan.

point-on-line incidences, $\Pi_C = \{d(p_1, p_0) = d_0, d(p_1, p_2) = d_1, d(p_2, p_3) = d_2, d(p_0, p_3) = d_3, onPL(p_0, l), onPL(p_2, l), onPL(p_3, l)\}$. The set of parameters includes the parameters related to the distances, $\Pi_P = \{d_0, d_1, d_2, d_3\}$. We consider as variant parameter $\lambda = d_2 = d(p_2, p_3)$.

Figure 4.9a shows the graph G of the geometric constraint problem $\Pi_\lambda =< \Pi_G, \Pi_C, \Pi_P >$. Figure 4.9b is the tree decomposition T_λ of the construction plan that solves the problem yielded by the solver. Figure 4.9c shows the associated h-graph $\mathcal{H}(G)$.

From $\mathcal{H}(G)$ we extract that Π_λ has three tree-decomposition steps: (l, p_2, p_3), (p_0, p_1, p_2) and (p_0, p_3, l). Tree-decomposition step (l, p_2, p_3) depends directly on λ for $p_2, p_3 \in (l, p_2, p_3)$, tree-decomposition step (p_0, p_1, p_2) depends indirectly on λ for it depends indirectly on (l, p_2, p_3), and finally tree-decomposition step (p_0, p_3, l) is independent on λ. Notice that in the tree-decomposition, the cluster with hinges (p_0, p_3, l) is decomposed after λ is fixed in a separated branch. This tree-decomposition step is not considered by the algorithm, and can always be constructed.

For the tree-decomposition step (l, p_2, p_3) the dictionary provides the critical values. The construction places point p_2 on the line through points p_0 and p_3 at a distance λ from p_0. Since we do not consider signed distances, the construction is clearly feasible for all λ with $0 \leq \lambda < \infty$.

Finally, we consider tree-decomposition step (p_0, p_1, p_2), which depends indirectly on the variant parameter λ. The hinges included in the cluster containing λ are p_0, p_2. Thus

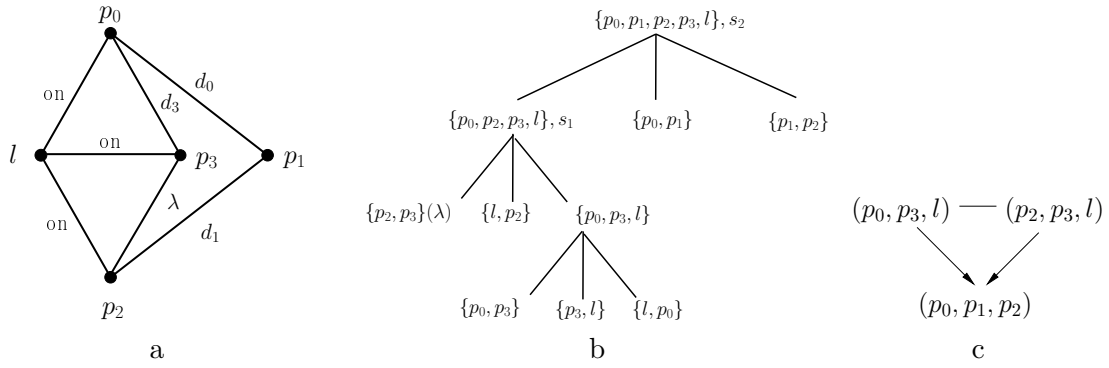


Figure 4.9: Piston and connecting rod crankshaft. a) Problem graph G. b) Tree-decomposition of the problem. c) Associated h-graph $\mathcal{H}(G)$.

the problem is transformed by replacing the constraint $\lambda = d(p_2, p_3)$ with $\mu = d(p_2, p_0)$.

The graph G' of the transformed problem Π_μ, a construction plan that solves it and the associated h-graph $\mathcal{H}(G')$ are shown from left to right in Figure 4.10b and Figure 4.10c.

In the transformed problem, tree-decomposition step (p_0, p_2, l) depends directly on μ. According to the dictionary of critical points, Π_μ is feasible if

$$|d_1 - d_0| \leq \mu \leq |d_1| + |d_0|.$$

Considering, for example, specific parameter values

$$d_0 = 5 \qquad d_1 = 8 \qquad d_3 = 14,$$

the transformed problem is feasible if $3 \leq \mu \leq 13$.

Figure 4.11 shows a construction plan and a construction of a solution to the modified problem. The index associated with μ is $I_\mu = \{s_1\}$, and, in general, there will be two different possible placements for point p_2, say p_2 and p_2', corresponding to the two possible intersections of circle c_0 with line l, see Figure 4.11b. Thus there are two different measures for the variant parameter λ for each critical value of μ.

Figure 4.12a shows the construction of the solution to the transformed problem at parameter value $\mu = 3$. Figure 4.12b shows the construction of the solution to the modified problem for the value $\mu = 13$. All possible signs assignments are depicted in both figures: p_2^+ represents the positive index assignment and p_2^- the negative one. Values measured for λ in $\mathrm{T}_\mu(13)$ are 1 and 27. Measures for λ taken in $\mathrm{T}_\mu(3)$ are 11 and 17. Therefore, the set of sorted critical values for the variant parameter λ is $\{1, 11, 17, 27\}$.

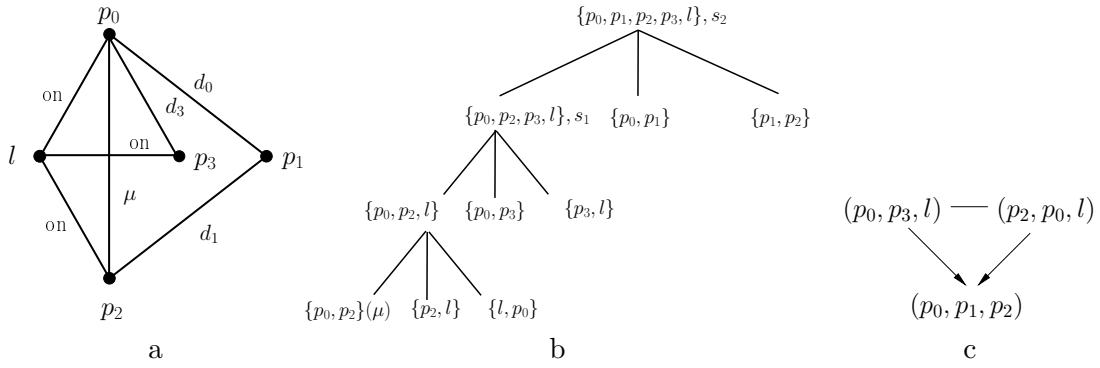


Figure 4.10: Piston and connecting rod crankshaft. Transformed problem. a) Problem graph. b) Decomposition tree of the problem. c) Associated h-graph $\mathcal{H}(G')$.

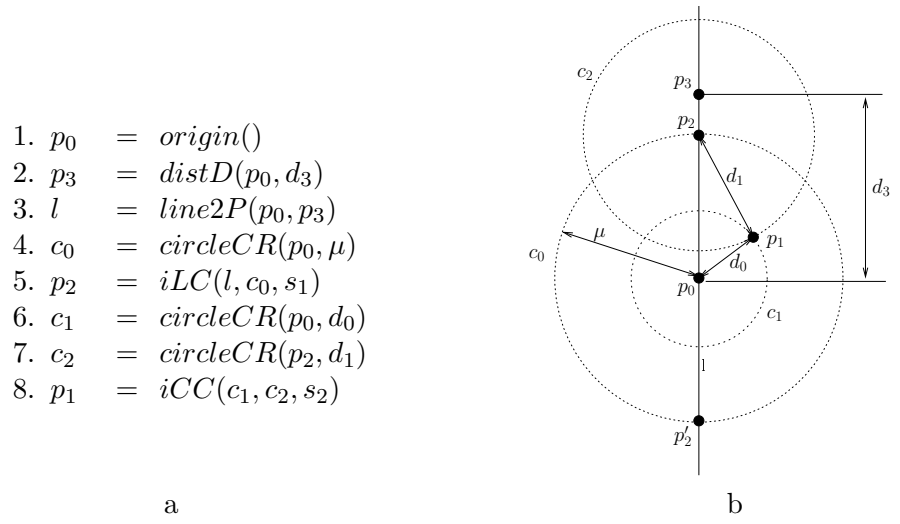


Figure 4.11: Piston and connecting rod crankshaft. Transformed problem. a) Construction plan. b) Geometric realization.

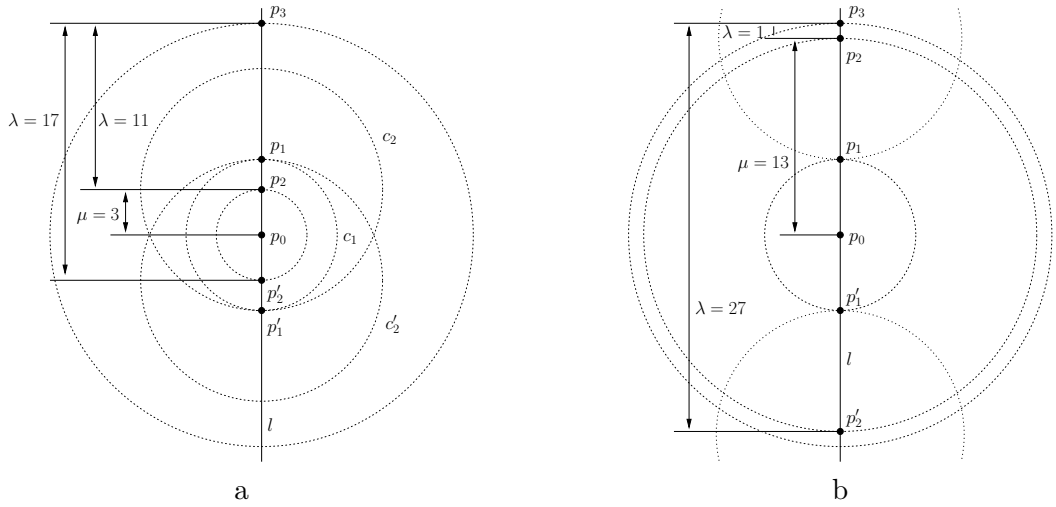


Figure 4.12: Construction of the transformed problem. a) Construction at $\mu = 3$. b) Construction at $\mu = 13$.

The index of the construction plan shown in Figure 4.8 that solves the piston and connecting rod crankshaft problem includes two signs $I = \{s_1, s_2\}$, hence up to four different intended solution instances can be selected. Concretely, for the index assignment $\sigma = \{s_1 = +1, s_2 = +1\}$, we check feasibility of T_λ at, say, λ taking values in $\{0.5, 5, 14, 22, 28\}$. We find out that the construction plan is feasible at critical values 1, 11, 17 and 27 and at the intermediate points 5 and 22, see Figure 4.13. Therefore the domain for the problem Π_λ and the index assignment σ is $[1, 11] \cup [17, 27]$.

Applying the same procedure to each of the four possible index assignments yields the feasibility domain depicted in Figure 4.14, where feasible intervals are filled in black. Notice that for signs assignments $s_1 = +1, s_2 = -1$ and $s_1 = -1, s_2 = -1$ the construction plan is unfeasible for any value of the variant parameter d_2.

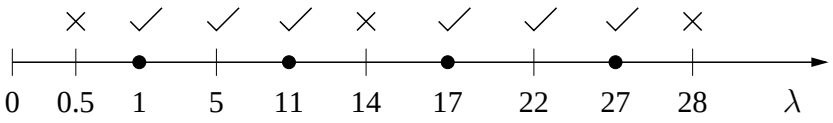


Figure 4.13: Feasibility for the piston and connecting rod cranckshaft problem. • represent critical points. | represent intermediate λ values. ✓ means that the construction plan is feasible. × means that the construction plan is unfeasible.

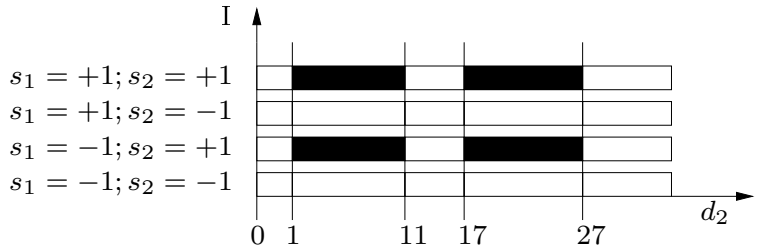


Figure 4.14: Piston and connecting rod crankshaft. Feasible domain for the variant parameter $\lambda = d_2$.

4.6 Conclusions

In this chapter, we have presented a complete proof of the method introduced by van der Meiden *et al.* in [103]. This method is based on the fact that different problems which only differ in the placement of the variant parameter have the same set of solution instances, which has been also proved. H-graphs have been used to determine efficiently the kind of dependence of each tree-decomposition step of the graph with respect to the variant parameter.

The main drawback of this method comes from the necessity of analyzing and constructing a new geometric constraint problem for each involved indirectly dependent generation step. This is a disadvantage for two different reasons. Firstly, to analyze and construct a problem is a very expensive process which increases the execution time of the method dramatically. Secondly, the more problems to analyze the method has, the more probability there is to find a non-decomposable problem.

A simple procedure which may decrease the order of the modified problem and improve the efficiency of the method when analyzing and constructing the transformed problems could be considered. In the worst case, the modified graph would remain the same.

The aim of changing the variant parameter is to measure the original variant parameter range values in the modified graph constructed at the parameter values where its feasibility changes. Then, the only necessary elements to be constructed in the second step of the method are the ones upon which the new variant parameter is defined, and the ones upon which the original variant parameter was defined. The rest of geometric elements are irrelevant for the measurement of the original variant parameter.

Consider now the original problem Π, represented by the graph $G = (V, E)$, and let the associated h-graph be $\mathcal{H}(G) = (\mathcal{V}, E_D, E_S)$. Assume that the original variant parameter is $\lambda = (u, v)$, which must be replaced with $\mu = (u', v')$. Using the h-graph $\mathcal{H}(G)$ and a

variation of the procedure in Section 3.5, Chapter 3, we can find the minimum complete subgraph of $\mathcal{H}(G)$ spanned by the geometric elements u, v, u', v'. Call this subgraph M. Then, we can apply the van der Meiden method to M, change the variant parameter λ for μ in M, solve M and then measure in it the value of λ. Since clearly $|V(M)| \leq |V|$, to solve M is less expensive in time than to solve G.

Analysis of the performance of this new approach would be necessary in order to determine the actual improvement with respect to the original method.

CHAPTER 5

The reachability problem

"Begin at the beginning," the King said, very gravely, "and go on till you come to the end: then stop."

Lewis Carroll, Alice in Wonderland

Reachability is a fundamental problem in the context of many models and abstractions which describe various computational processes. Analysis of the computational traces and predictability questions for such models can be formalized as a set of different reachability problems. In general reachability can be formulated as follows: given a computational system with a set of allowed transformations, also called functions, decide whether a certain state of a system is reachable from a given initial state by the allowed transformations. We present in this chapter an approach which solves the reachability problem for geometric constraint problems with one degree of freedom. The results of this work have been published in Hidalgo and Joan-Arinyo, [39, 40].

5.1 Continuity and continuous transitions

In this section we properly define some basic concepts which will be used all along this chapter. The first one is the concept of continuity in geometric constructions, which is a key point in the definition of the reachability problem itself. We address also the concept of transition in the domain of a geometric problem, which is the essential concept of our approach. Transitions will allow continuity at the critical values, in contrast to other techniques that avoid them, and will be used as connections between different parts of the reachability problem domain.

5.1.1 Continuity

The key concepts in dynamic geometry are interaction and change. If the value assigned to the variant parameter, defined in Section 2.3, is interactively changed, the user expects the whole construction to follow. Moreover, whenever the variant parameter moves along continuous paths, the user expects that the geometric elements in the construction move along continuous paths as well. Since this is not always the case, we need to properly formalize this concept.

Continuity and behavior are two different concepts that appear to be tightly related in dynamic geometry. Since a naive application of the mathematical continuity notion leads to unexpected or unwanted dynamic behavior, we propose to clearly decouple them and give a specific definition for behaviour, Hidalgo and Joan-Arinyo [38].

Following Denner-Broser, [15], and Richter-Gerbert *et al.*, [90], we first introduce the concept of *variant parameter path.*

Definition 5.1.1
Let $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ *be a geometric constraint problem with one variant parameter* λ *and let* $T(\sigma, \lambda)$ *be a construction plan that solves* Π. *With the variant parameter* λ *we associate a continuous path* $\lambda(t) : [0, 1] \rightarrow \mathbb{R}^+$*, called variant parameter path.*

A variant parameter path is just a path followed by the variant parameter in $\mathbb{R}^+$.

Now we define the concept of dynamic evaluation under a movement given by a variant parameter path.

Definition 5.1.2
Let $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ *be a geometric constraint problem with one variant parameter* λ *and let* $T(\sigma, \lambda)$ *be a construction plan that solves* Π. *Let* $n = |\sigma|$ *be the number of sings in the index* σ *and* $\lambda(t)$ *a variant parameter path. A dynamic evaluation of the construction*

plan T under the path $\lambda(t)$ is an assignment of functions

$$\sigma_i(t) : [0,1] \to \{s\}^n, \quad \textit{with} \quad s \in \{-1,+1\}$$

such that for all $t \in [0,1]$, $T(\sigma_i, \lambda)(t) = T(\sigma_i(t), \lambda(t))$ is a solution instance to the problem.

We build our approach on top of a ruler-and-compass solver, that is to say, we solve equations with degree at most two. Therefore, construction plans include additions, differences, products, divisions and square root operations. In this scenario, critical values or discontinuities can appear when trying to divide by zero or computing the square root of a value equal or lower than zero. However, since operations in a construction plan $T(\sigma, \lambda)$ are continuous, a dynamic evaluation of T makes geometric elements to move along continuous paths, as long as the variant parameter path $\lambda(t)$ does not go through a critical value.

Dynamic evaluations are called *continuous* or *continuous evaluations* if they are continuous for all $t \in [0,1]$ in the usual way. That means that the position function of each geometric element is continuous for all $t \in [0,1]$.

5.1.2 Continuous transitions

As said in Section 2.3, critical values are values of the variant parameter for which the feasibility of the corresponding construction change. Critical values thus are endpoints of the domain intervals of the problem. Critical points are defined as the solution instances constructed at the critical values of the problem, that is, constructions of the problem at the endpoints of the domain intervals.

As mentioned in the previous section, we only work with ruler-and-compass problems, which can be defined by degree two equations. Quadratic equations represent conics on the plane. A conic is a function with two different solutions at every point of its domain except at the endpoints, where both solutions converge in a single one. Although the functions with which we work are combination of quadratic equations and in general more complicated as simple conics, many times they have two different solutions which converge at the endpoints of the domain intervals. That means that the critical points at critical values with different index assignments may converge into the same construction instances. We refer to such points as *degenerate configurations*.

Our approach to the geometric constraint solving relies on the fact that degenerate configurations represent the opportunity to perform an index assignment change in such a way that the dynamic evaluation of the construction plan is continuous. Far from avoiding this kind of configurations, our method uses them in its own benefit.

To formally define concepts about degenerate configurations and deal with critical val-

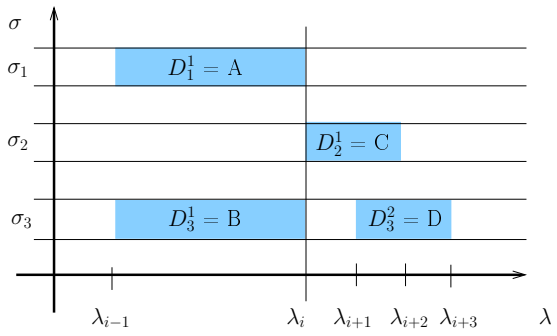


Figure 5.1: Domain intervals of the domain of a geometric constraint problem.

ues in the continuous evaluation of a construction plan we introduce the concept of *transition*.

Definition 5.1.3
Let $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ *be a geometric constraint problem with one variant parameter* λ *and let* $T(\sigma, \lambda)$ *be a construction plan that solves* Π. *Let* λ_i *be a critical variant parameter value of* Π. *Let* σ_j *and* σ_k *be two index assignments such that* $I_j^i = T(\sigma_j, \lambda_i)$ *and* $I_k^i = T(\sigma_k, \lambda_i)$ *are two solution instances of the construction plan* T. *Then we say that the pair of instances* (I_j^i, I_k^i) *define a transition in the domain of* Π.

Filled cells in Figure 5.1 are examples of domain intervals.

Since critical variant parameter values represent the bounds of the domain intervals of the problem, transitions are always defined between the endpoints of the domain intervals of the problem. Assuming that the domain intervals in Figure 5.1 are closed in the critical variant parameter λ_i, we can identify three transitions: (I_1^i, I_2^i), (I_1^i, I_3^i) and (I_3^i, I_2^i).

We will consider two different types of continuous transition. Notice that, for angle type variant parameters, we have considered that angles are defined inside the interval $[-\pi/2, \pi/2]$, and we have identified angles $-\pi/2$ and $\pi/2$ modulo π by assigning a sign to the angle. Under such identification, we can consider a transition defined between instances $T(\sigma_i, -\pi/2)$ and $T(\sigma_j, \pi/2)$. Such transitions shall be called *improper transitions*, in opposition to transitions between instances with the same critical value, which shall be called *proper transitions*.

Among the combinatorial number of possible transitions, our interest focuses on those which assure the continuity of the dynamic evaluations under paths traversing them.

Definition 5.1.4
Let $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ *be a geometric constraint problem with one variant parameter* λ

and let $T(\sigma, \lambda)$ be a construction plan that solves Π. Let λ_i and λ_j be two critical variant parameter values. The transition (I_k^i, I_l^j) is called continuous if the solution instances I_k^i, I_l^j are congruent modulo rigid translations and rotations.

We will also consider two different types of continuous transitions. A *proper continuous transition* will indicate that a proper transition is continuous. An *improper continuous transition* will indicate that an improper transition is continuous.

Whether proper or improper, continuous transitions are at the core of our approach to solve the reachability problem, for they allow to change the index assignment in a way such that the dynamic evaluation of the construction plan under variant parameter paths is continuous. Continuous transitions are given their name because of the following two properties.

Lemma 5.1.1
Let $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ be a geometric constraint problem with one variant parameter λ and let $T(\sigma, \lambda)$ be a construction plan that solves Π. Let $\lambda(t)$, with $t \in [0, 1]$, be a variant parameter path with just one critical value λ_c in [0,1]. Let (I_k^c, I_l^c) be a continuous transition at λ_c. Then, $T(\sigma, \lambda)(t)$ is continuous.

Proof
Once σ and λ are fixed, the instance generated by the construction plan $T(\sigma, \lambda)$ is unique up to rigid translations and rotations. □

This result is easily extended to a path with a finite number of critical values. The following theorem is then straightforward.

Theorem 5.1.2
Let $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ be a geometric constraint problem with one variant parameter λ and let $T(\sigma, \lambda)$ be a construction plan that solves Π. Let $\lambda_i, 1 \leq i \leq n$ be the finite set of critical values in the path $\lambda(t)$. Then a dynamic evaluation of $T(\sigma, \lambda)(t)$ for the path $\lambda(t)$ is continuous if there is at least one continuous transition at every critical variant parameter value λ_i.

Proof
Just apply Lemma 5.1.1 to each critical value in the path. Since the number of critical values is finite, this can be always done. □

From now on we will just consider continuous transitions, and we shall refer to them as transitions. We shall refer to proper continuous transitions simply as *proper transitions*, and to improper continuous transitions simply as *improper transitions*.

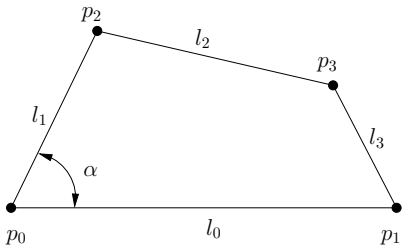


Figure 5.2: Four-bars linkage problem scheme.

5.1.3 Case study: the four-bars linkage

To illustrate the continuous transition concept we develop a complete case study. We consider the well known four-bar linkage shown in Figure 5.2. We compute the domain and corresponding transitions depending on different values assigned to the bars lengths.

The four-bars linkage problem is composed by four points or joins and four bars with a given length. Consider the points are given names p_0, p_1, p_2 and p_3, the bars are given names l_0, l_1, l_2 and l_3 and its lengths are d_0, d_1, d_2 and d_3, respectively. Bars l_0 and l_1 are related by an angular constraint α.

The domain of the problem depends on the relations existing among the values of the bars' lengths. We analyze the problem domain for each possible relation between bars, assuming that the lengths are fixed but arbitrary, and the variant parameter is the angle α.

Because of the nature of the problem, for every solution instance I with parameter value α_0 there exists an index assignment σ such that the solution instance $T(\sigma, -\alpha_0)$ is the symmetric configuration of I with respect to the bar with length l_0. Thus, if α_0 belongs to the domain of the problem, also $-\alpha_0$ belongs to it. Then, all the domain intervals are symmetric with respect to the value 0 and the inclusion or exclusion of this point will change the shape of the domain.

Since our system considers angles defined within the interval $[-\pi/2, \pi/2]$, 0 and π represent the same angle and the inclusion or exclusion of the latter one will also change the shape of the domain. Besides, the inclusion of points $\pi/2$ and $-\pi/2$ in the domain allows the presence of improper transitions between the domain intervals.

Therefore, there are three values for angle α which determine the shape of the domain of the problem: 0, $\pi/2$ and π. The condition to assure the inclusion of those values in the domain is that the distance between points p_1, p_2, once the angle has been fixed, allows the construction of point p_3, see Figure 5.3. Then, points p_1, p_2, p_3 must fulfill the triangular

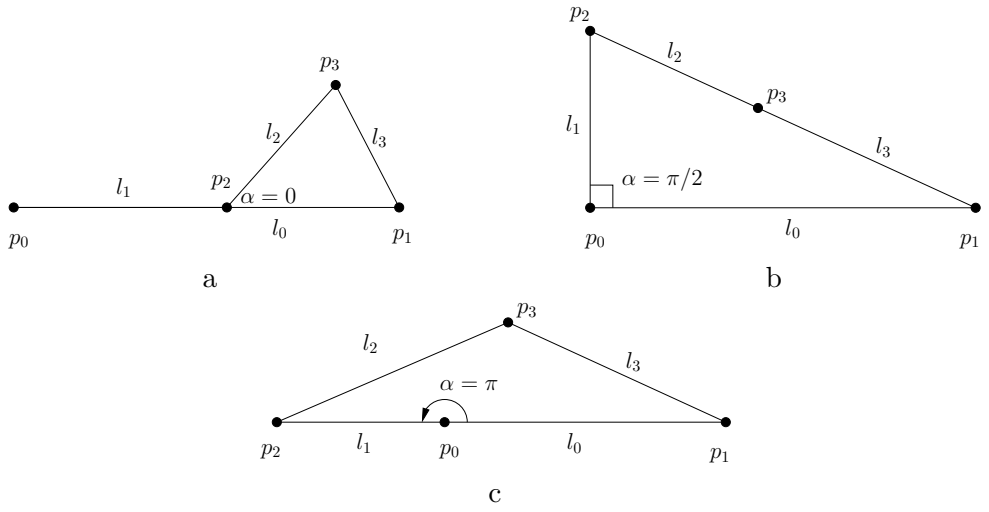


Figure 5.3: Instances of the four-bars linkage. a) The four-bars linkage for a value of $\alpha = 0$. b) The four-bars linkage for a value of $\alpha = \pi/2$. c) The four-bars linkage for a value of $\alpha = \pi$.

inequality.

- $\alpha = 0$: In this case, point p_2 must be on bar l_0, see Figure 5.3a. The distance between points p_1 and p_2 is $|d_0 - d_1|$. Thus, the condition for $\alpha = 0$ to belong to the domain is

$$|d_3 - d_2| \leq |d_0 - d_1| \leq d_2 + d_3$$

- $\alpha = \pi/2$: In this case, the distance between points p_1 and p_2 is $\sqrt{d_0^2 + d_1^2}$, see Figure 5.3b. Then, the condition for $\alpha = \pi/2$ to belong to the domain is

$$|d_3 - d_2| \leq \sqrt{d_0^2 + d_1^2} \leq d_2 + d_3$$

- $\alpha = \pi$: In this case, points p_0, p_1, p_2 are collinear, see Figure 5.3c. The distance between points p_1 and p_2 is $d_0 + d_1$. Therefore the condition for $\alpha = \pi$ to belong to the domain is

$$|d_3 - d_2| \leq d_0 + d_1 \leq d_2 + d_3$$

The different combinations of exclusion and inclusion of these three angle values in the domain of the four-bars linkage problem give rise to $2^3 = 8$ combinations, leading to 8

different domain shapes. However, we will see that one of them can not take ever place. Consider that the domain includes 0 and π. Then, the relations

$$|d_3 - d_2| \leq |d_0 - d_1| \leq d_2 + d_3$$

$$|d_3 - d_2| \leq d_0 + d_1 \leq d_2 + d_3$$

should hold. But it is also a straightforward computation to see that

$$|d_0 - d_1| \leq \sqrt{d_0^2 + d_1^2} \leq d_0 + d_1$$

which easily yields

$$|d_3 - d_2| \leq |d_0 - d_1| \leq \sqrt{d_0^2 + d_1^2} \leq d_0 + d_1 \leq d_2 + d_3.$$

Then, the condition for angle $\alpha = \pi/2$ to belong to the domain is fulfilled. That is, whenever angles 0 and π are included in the domain, also angle $\pi/2$ is included. The case in which angles 0 and π are included in the domain but angle $\pi/2$ is not is destined to a contradiction. The number of possible domain shapes is then 7.

We analyze now the domain shape of each one of the seven different cases resulting from the inclusion or exclusion of the angle values 0, π and $\pi/2$. Solution instances for endpoints of the domain intervals as well as their respective domain shapes are shown in Figures 5.4 through 5.10. Continuous transitions among the domain intervals are represented by arrows.

Case 1: $0 \in \mathrm{D}(\alpha)$, $\pi/2 \notin \mathrm{D}(\alpha)$, $\pi \notin \mathrm{D}(\alpha)$. Figure 5.4 shows the problems domain and the solution instance for which the variant parameter α is the upper bound of the domain, α_0. A proper transition at this value allows the change of index assignment to place point p_3. The symmetrical construction with respect to l_0 represents the solution instance for which the variant parameter is $-\alpha_0$.

Case 2: $0 \in \mathrm{D}(\alpha)$, $\pi/2 \in \mathrm{D}(\alpha)$, $\pi \notin \mathrm{D}(\alpha)$. Figure 5.5 shows the problem domain and the solution instance for which the variant parameter α is the upper bound of the domain, α_0 where $\pi > \alpha_0 > \pi/2$. However, our system deals with angle ranges from $-\pi/2$ to $\pi/2$. We consider then every angle modulo π. The value of the variant parameter for this instance is then $\alpha_0 - \pi \in [-\pi/2, \pi/2]$. There are proper transitions at this point and at the symmetric one, and improper transitions between some of the domain intervals.

Case 3: $0 \in \mathrm{D}(\alpha)$, $\pi/2 \in \mathrm{D}(\alpha)$, $\pi \in \mathrm{D}(\alpha)$. Figure 5.6 shows the solution instance for $\alpha = \pi$ and the problem domain. Improper transitions exist between some of the domain intervals. Notice that there are two different connected components in the domain. This is due to the fact that there is no way to change the index assigned to the placement of point p_3 in such a way that the resulting dynamic evaluation is continuous.

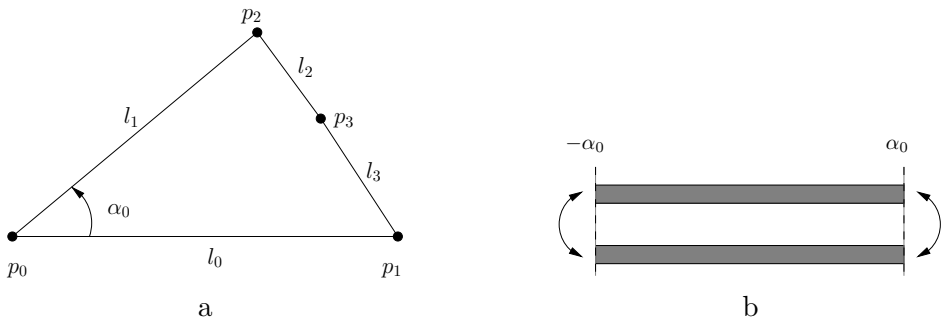


Figure 5.4: Case 1. a) Solution instance at value $\alpha = \alpha_0$. b) Domain of the problem.

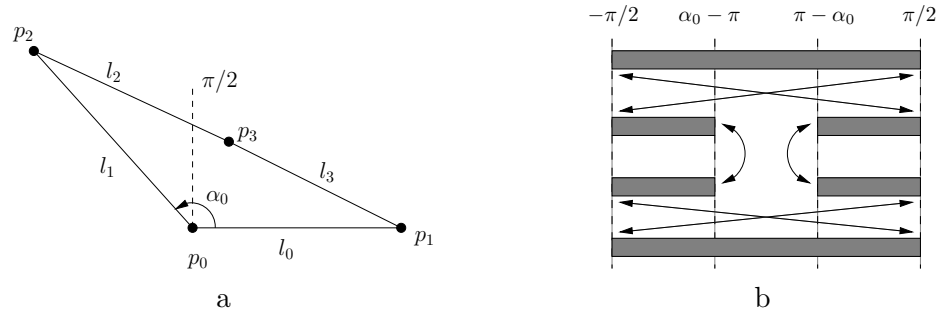


Figure 5.5: Case 2. a) Solution instance at value $\alpha = \alpha_0$. b) Domain of the problem.

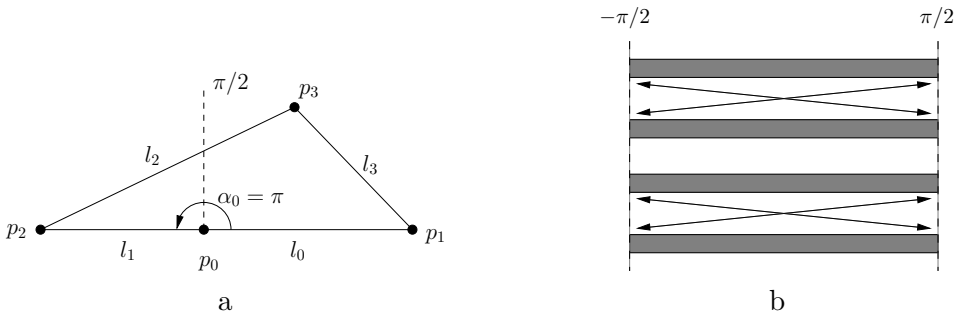


Figure 5.6: Case 3. a) Solution instance at value $\alpha = \alpha_0$. b) Domain of the problem.

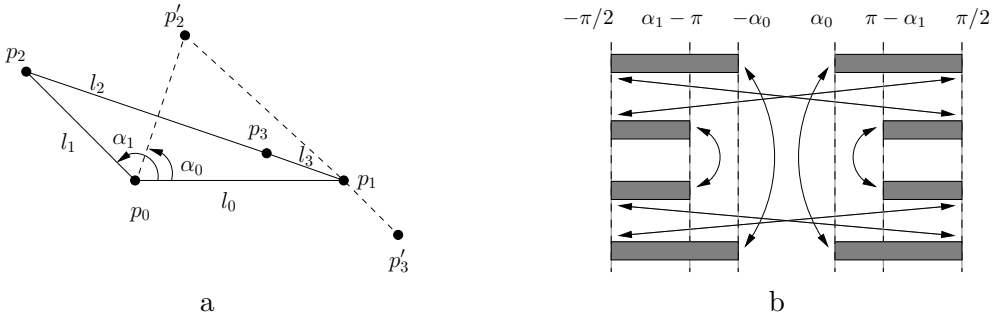


Figure 5.7: Case 4. a) Solution instance at value $\alpha = \alpha_0$. b) Domain of the problem.

Case 4: $0 \notin \mathrm{D}(\alpha)$, $\pi/2 \in \mathrm{D}(\alpha)$, $\pi \notin \mathrm{D}(\alpha)$. Figure 5.7 shows the solution instance for which the variant parameter α is the upper bound of the domain, α_1, and the problem domain. In dashed lines is depicted the instance for which the variant parameter is the lower bound, α_0. The domain is defined between both angles, with the particularity that value α_1 is considered modulo π. There are proper transitions at these points and at the symmetric ones, and improper transitions between some of the intervals.

Case 5: $0 \notin \mathrm{D}(\alpha)$, $\pi/2 \notin \mathrm{D}(\alpha)$, $\pi \notin \mathrm{D}(\alpha)$. Figure 5.8 shows the solution instance for which the variant parameter α is the upper bound of the domain, α_1, and the domain of the problem. In dashed lines is depicted the instance for which the variant parameter is the lower bound, α_0. There are proper transitions at these points and at the symmetric ones. In this case the domain is made of two separated connected components. In case $\alpha_0, \alpha_1 > \pi/2$ the domain's shape will remain the same, as we will consider both angles modulo π.

Case 6: $0 \notin \mathrm{D}(\alpha)$, $\pi/2 \in \mathrm{D}(\alpha)$, $\pi \in \mathrm{D}(\alpha)$. Figure 5.9 shows the solution instance for which the variant parameter α is the upper bound of the domain, α_1 and the problem domain.

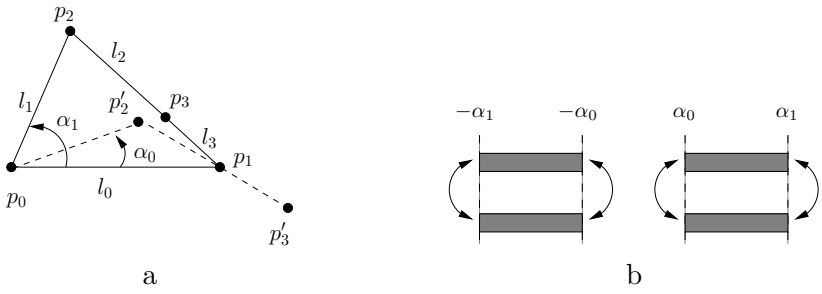


Figure 5.8: Case 5. a) Solution instance at value $\alpha = \alpha_0$. b) Domain of the problem.

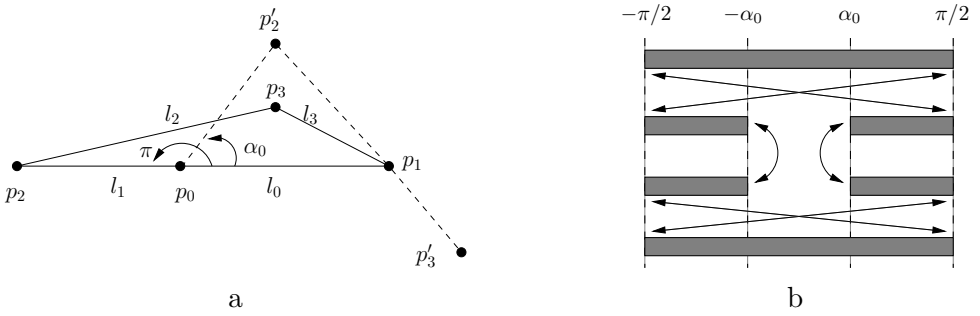


Figure 5.9: Case 6. a) Solution instance at value $\alpha = \alpha_0$. b) Domain of the problem.

In dashed lines is depicted the instance for which the variant parameter is the lower bound, α_0. Since $\alpha_0 < \pi/2$, its value modulo π is itself. There are proper transitions at this point and at the symmetric one, and improper transitions between some of the intervals.

Case 7: $0 \notin \mathrm{D}(\alpha)$, $\pi/2 \notin \mathrm{D}(\alpha)$, $\pi \in \mathrm{D}(\alpha)$. Figure 5.10 shows the solution instance for which the variant parameter $\alpha = \pi$, and the domain of the problem. In dashed lines is depicted the instance for which the variant parameter is the lower bound of the domain, α_0. Proper transitions exist at this point. Notice that this gives rise to the same domain shape as in Case 1. The reason is that Case 1 domain contains angle 0 whereas Case 7 contains angle π, and none contains angle $\pi/2$. Angle π is represented by angle 0. However, the index assignments for feasible constructions will be exactly the opposite in both cases.

The seven cases analyzed lead to all the possible shapes of the domain for the four-bars linkage. Now we present some particular cases of parameter value assignments in this problem which highlight some features of our theory.

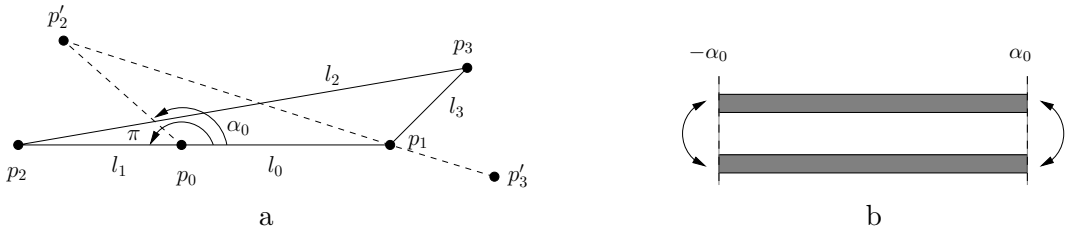


Figure 5.10: Case 7. a) Solution instance at value $\alpha = \alpha_0$. b) Domain of the problem.

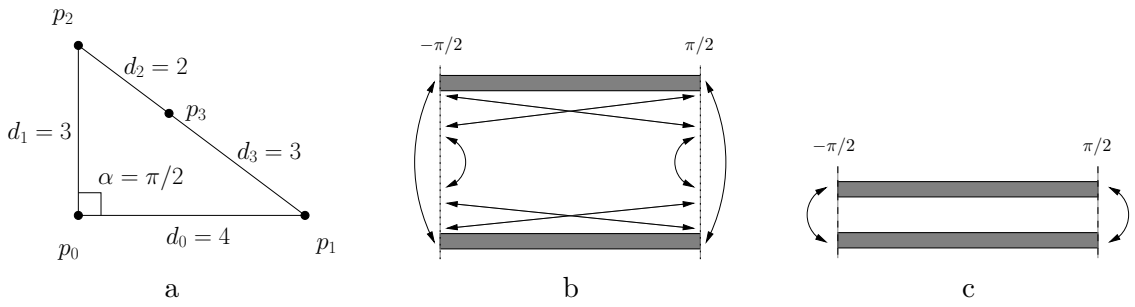


Figure 5.11: Particular case of the four-bars problem. a) Configuration for $\alpha = \pi/2$. b) Domain. c) Simplified domain.

Consider the case where $\sqrt{d_0^2 + d_1^2} = d_2 + d_3$. Figure 5.11a shows a particular instance for the variant parameter value $\alpha = \pi/2$ and bar lengths $d_0 = 4, d_1 = 3, d_2 = 2, d_3 = 3$ which fulfill the condition. As seen in the picture, angle value $\alpha = \pi/2$ is an endpoint of the domain of this problem. This is then a particular case in the common boundary of cases 1 and 2. In fact, it can be seen as a particular case of Case 1 for which the maximum angle α_0 is increased until reaching value $\pi/2$, as well as a particular case of Case 2 for which the maximum angle α_0 is decreased until reaching value $\pi/2$.

When computing the domain of this problem with the method described in Chapter 4, the result includes four intervals with zero measure, which we shall call *improper*, defined as $[-\pi/2, -\pi/2]$ and $[\pi/2, \pi/2]$. The whole computed domain together with the arising continuous transitions is shown in Figure 5.11b. These four domain intervals do not provide any information on the domain and are absolutely useless for they have only one instance and no interior path to follow. Transitions involving them can be simplified just by removing them. Despite the fact that they are theoretically valid, the decision to eliminate them has been taken in favor of a major clarity and simplicity. The resulting domain is shown in Figure 5.11c.

There are other domain configurations in the boundary of two of the different cases described above, which arise when at least one of the endpoint angles takes value 0, π or $\pi/2$. In most of them similar cases of improper intervals appear. They are all treated in the same way.

Independently of the fact that the bar lengths fulfill Pythagoras' Theorem, the problem in Figure 5.11a presents another singular characteristic. Effectively, for this singular problem, $\alpha = 0°$ is a critical value which happens not to be a bound of any domain interval since feasibility is achieved at both sides of it. However, at this point there are two equal solution instances for the two different index assignments. The solution instance of the problem for this angle value is shown in Figure 5.12a. We consider that intervals should

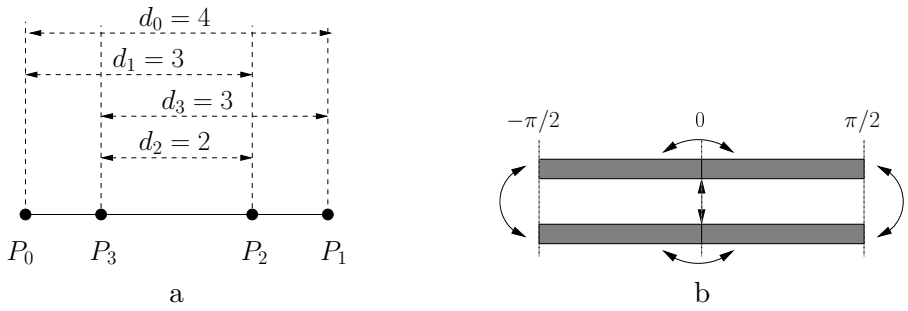


Figure 5.12: Particular case. a) Instance of the problem in Figure 5.11a at variant parameter value $\alpha = 0$. b) Split domain.

be split at the connecting point, in this way continuous transitions are only defined at endpoints of the domain intervals, and not at internal points. Figure 5.12b shows the final domain of this problem, once each interval is split by the continuous transition point.

It is usual that specific behaviours appear whenever bar lengths define implicit theorems. The most common case arises when values assigned to the bar lengths are coincident. Problems with parameters defining implicit theorems are not always well-constrained, because some of the constraints may become redundant. Identifying and dealing with these scenarios are still open problems in Geometric Constraint Solving. However, in this concrete problem well-constraintness is not affected by the fact that some bars have coincident lengths.

As a final example we consider the case in which $d_0 = d_2 = 5$ and $d_1 = d_3 = 4$. Figure 5.13 shows the corresponding domain where up to 16 continuous transitions can be identified.

5.2 An algorithm for the reachability problem

In plain words, the reachability problem consists on deciding whether there exists a continuous way of transforming an initial given instance in a predefined final one. The key concept of the reachability problem is the continuity. We formally state the reachability problem we solve as follows (see also Denner-Broser [16]).

> Let $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ be a geometric constraint problem with one variant parameter λ and let $T(\sigma, \lambda)$ be a construction plan that solves Π. Let $I_s = T(\lambda_s, \sigma_s)$ and $I_e = T(\lambda_e, \sigma_e)$ be respectively a starting and ending instance of

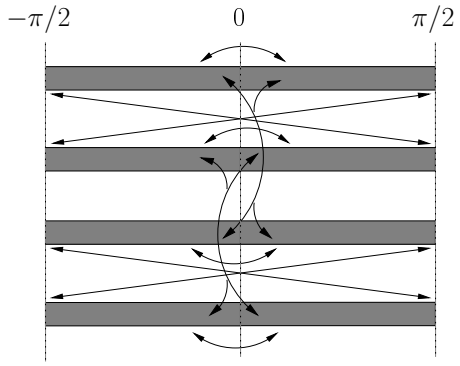


Figure 5.13: Domain for the fourbars problem with $d_0 = d_2 = 5$ and $d_1 = d_3 = 4$.

T. Decide whether there is a continuous path $\lambda(t) : [0, 1] \to \mathbb{R}^+$ of the variant parameter and assignments of index

$$\sigma(t) : [0, 1] \to \{-1, +1\}^n$$

for which there is a corresponding continuous evaluation $T(\sigma, \lambda)(t)$ from I_s to I_e, that is, such that $T(\sigma, \lambda)(0) = I_s$ and $T(\sigma, \lambda)(1) = I_e$.

We present in this section our approach to solve the reachability problem. Consider a geometric constraint problem with one variant parameter and a reachability problem defined on it. The proposed technique has three steps. First the set of points where the geometric construction is not feasible and the domain of the variant parameter with respect to the geometric constraint problem at hand are computed. Then transitions at these points are captured as a graph. Finally the reachability is decided by searching in this graph a path from the starting geometric construction to the ending one. The search is performed by applying the A* algorithm.

As a proof of concept, we have implemented the approach in the context of our dynamic geometry system based on constructive geometric constraint solving, Hidalgo *et al.*, [39, 40]. Preliminary results prove that the approach is both effective and efficient from a practical point of view. We shall explain in detail the process all along this section.

5.2.1 The transitions graph

We present in this section the graph capturing the different transitions among the domain intervals of the domain of a geometric constraint problem.

Consider a geometric constraint problem $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ with one variant parameter λ and let $T(\sigma, \lambda)$ be a construction plan that solves Π. We define now the *transitions graph* as follows.

Definition 5.2.1
A transitions graph is a graph $TG = (V, E_I, E_T)$, where vertices in V represent specific instances of the problem associated to the endpoints of the domain intervals by means of a pair (λ_i, σ_j), where λ_i is a critical value and σ_j an index assignment. Edges $((\lambda_i, \sigma_j), (\lambda_k, \sigma_l))$ in E_I represent intervals which begin at (λ_i, σ_j) and end at (λ_k, σ_l). Edges in E_T represent proper and improper transitions occurring between the instances in V.

Each edge in E_I represents an interval and are labeled with the name of this interval. An edge in E_I from the instance (σ_1, λ_1) to the instance (σ_2, λ_2) is by $d = |\lambda_1 - \lambda_2|$, since this is the distance the path must follow to reach the following vertex.

Consider that $v_1 = (\lambda_1, \sigma)$ and $v_2 = (\lambda_2, \sigma)$ are two vertices of the transitions graph representing two different instances I_1, I_2 with the same index assignment σ. Consider that there exists a domain interval A bounded by λ_1 and λ_2, with $\lambda_1 < \lambda_2$, whose first and last instances are I_1, I_2, respectively. Then edge $e = (v_1, v_2) \in E_I$ is labeled with the letter A and $w(e) = |\lambda_1 - \lambda_2|$.

Each edge in E_T represents a transition among two domain intervals and is labeled with the $\emptyset$ symbol. All edges in E_T have an associated weight of 0. These edges represent the idea that no path must be followed in order to reach the following vertex.

Consider that $v_1 = (\lambda_1, \sigma)$ and $v_2 = (\lambda_2, \sigma)$ are two vertices of the transitions graph representing two different instances I_1, I_2 and $\lambda_1 = \lambda_2 \mod \pi$. Consider that there exists a continuous transition between I_1 and I_2. Then edge $e = (v_1, v_2) \in E_T$ is labeled with the symbol $\emptyset$ and $w(e) = 0$.

To compute the transitions graph we assume that the domain is given as a bucket sort with as many buckets as different critical variant parameter values are bounds of a domain interval. Each bucket includes the solution instances at the corresponding critical value which bounds of a domain interval. We call them *bounding instances*. Each bounding instance related to a bound λ within a bucket stores: the related index assignment σ, the solution instance, $I = T(\sigma, \lambda)$, and the name of the domain interval of which is bound, plus a flag in the set $\{l, u\}$ that identifies whether the critical value corresponds to the lower bound or to the upper bound of the domain interval. Whenever a domain interval is open at a critical value, the construction plan instance points to nil. We call this representation of the domain the *bucket sorted domain*, BSD in short. For the domain example in Figure 5.1, the bucket sorted domain would be the one shown in Figure 5.14.

We also consider a simple structure which stores, related to each domain interval, the node of the transitions graph corresponding to its lower bounding instance. This table

$$
\begin{aligned}
\lambda_{i-1} &\rightarrow (\sigma_1, I_1^{i-1}, A, l),\ (\sigma_3, I_3^{i-1}, B, l) \\
\lambda_{i} &\rightarrow (\sigma_1, I_1^{i}, A, u),\ (\sigma_2, I_2^{i}, C, l),\ (\sigma_3, I_3^{i}, B, u) \\
\lambda_{i+1} &\rightarrow (\sigma_3, I_3^{i+1}, D, l) \\
\lambda_{i+2} &\rightarrow (\sigma_2, I_2^{i+2}, C, u) \\
\lambda_{i+3} &\rightarrow (\sigma_3, I_3^{i+3}, D, u)
\end{aligned}
$$

Figure 5.14: Domain represented as a bucket sort table of intervals.

represents the intervals which are active at a critical value, and we call it the *active interval list*, AIL in short.

The transitions graph is built applying a scan-line algorithm, [21]. The events that move the scan-line are the critical variant parameter values, λ_i. We create a vertex for each one of the bounding instances within a bucket. In case it represents the upper bounding instance of a domain interval, we use the table AIL to add an interval edge joining it to the vertex representing the lower bounding instance of the interval.

When all the bounding instances of a bucket have been added as vertices to the transitions graph, coincidence among them is tested in order to add the corresponding transitions edges. In case that the parameter is an angle and the considered bound is $\pi/2$, also improper transitions are checked.

Algorithms 7 through 9 show how we actually compute the transitions graph. A bounding instance B contains four self-explanatory fields: *index, instance, interval* and *flag*, standing for the four entries stored in it. We consider also the boolean function *congruent*, which returns TRUE in case that two instances are congruent modulo rigid transformations, and FALSE otherwise.

Figure 5.15 shows the transitions graph yielded by this algorithm when applied to the problem with domain depicted in Figure 5.1 and transitions between all the instances at values λ_{i-1} and λ_i. Notice that bounding instances of a given interval are joined by an interval edge, and transitions at bounds λ_{i-1} and λ_i are joined by a transition edge.

Figure 5.17 shows the transitions graph yielded by Algorithm 7 when applied to the problem with domain and transitions depicted in Figure 5.16. Notice that the graph has two disconnected components, therefore no continuous transitions between them can occur.

Algorithm 7 Computing the Transitions Graph

```
Input: BSD, the bucket sorted domain
Output: TG( V, E_T, E_I ), the transitions graph

V = ∅
E_T = ∅
E_I = ∅
for all λ_i in BSD do
  LB = Bounding Instances List( λ_i )
  for all B in LB do
    V_n = ( λ_i, B.index )
    V = V ∪ { V_n }
    Add Intervals Edges( B, V_n, E_I )
  end for
  Add Transitions Edges( λ_i, λ_i, E_T )
  if parameterType == ANGLE and λ_i == π/2 then
    Add Transitions Edges( −π/2, π/2, E_T )
  end if
end for
return ( V, E_T, E_I )
```

Algorithm 8 Add Intervals Edges

```
Input: B, a bounding instance,
       V_n, the vertex related to B,
       E_I, the set of intervals edges
Output: E_I, the updated set of intervals edges

if B.flag == 'l' then
  AIL.add( B.interval, V_n )
else
  V_j = AIL.getVertex( B.interval )
  E_I = E_I ∪ {( V_n, V_j )}
end if
return E_I
```

Algorithm 9 Add Transitions Edges

Input: λ_1, a critical value,
λ_2, a critical value,
E_T the set of transitions edges
Output: E_T the updated set of transitions edges

LB1 = Bounding Instances List(λ_1)
LB2 = Bounding Instances List(λ_2)
for all B_i in LB1 **do**
  for all B_j in LB2, $B_i \neq B_j$ **do**
    if congruent(B_i.instance, B_j.instance) **then**
      $E_T = E_T \cup \{(B_i, B_j)\}$
    end if
  end for
end for
return E_T

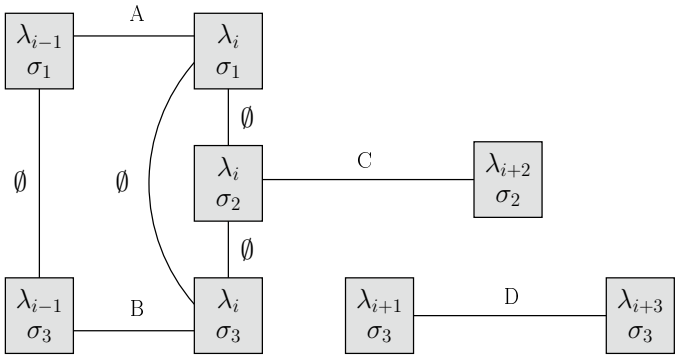


Figure 5.15: Transitions graph for the example in Figure 5.1.

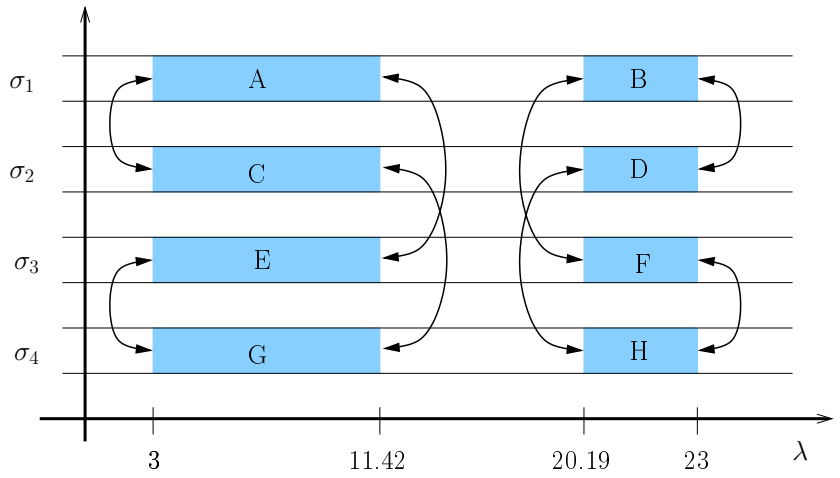


Figure 5.16: Domain and continuous transitions of a geometric problem. Continuous transitions are represented as arrows between endpoints of the domain intervals.

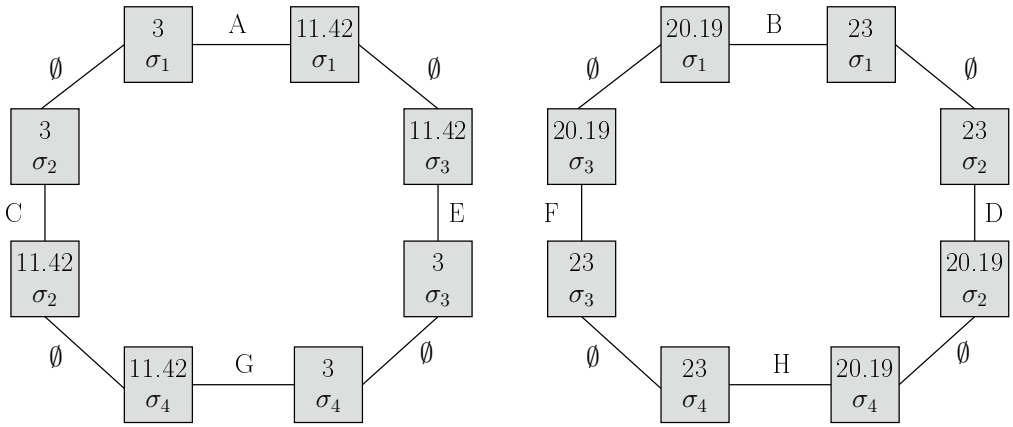


Figure 5.17: Transitions graph for the domain in Figure 5.16.

5.2.2 Deciding reachability

Consider a geometric constraint problem $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ with one variant parameter λ and let $T(\sigma, \lambda)$ be a construction plan that solves Π. Assume that $I_s = T(\sigma_s, \lambda_s)$ and $I_e = T(\sigma_e, \lambda_e)$ stand respectively for the starting and ending instances of a reachability problem stated over Π.

It is clear that the reachability problem can be positively solved if and only if solution instances I_s and I_e belong to domain intervals in the same connected component of the transitions graph. Assumed that I_s and I_e belong to the same connected component of the transitions graph, to find a continuous evaluation starting at I_s and ending at I_e, all what we need to do is first to identify the intervals to which I_s and I_e belong to. Then we need to search for the existence of a path connecting the corresponding domain intervals.

The method to solve reachability consists in searching in the transitions graph for a path between the starting and ending instances. In order to solve reachability problems involving starting and ending instances with variant parameters different to the endpoints of the intervals, vertices with these concrete values must be added to the transitions graph. We define therefore the extended transitions graph.

Definition 5.2.2
Let $TG = (V, E)$ be the transitions graph of a geometric problem Π, D an interval domain of Π and $I = T(\lambda, \sigma)$ a solution instance such that λ is not a critical value and $\lambda \in D$. Let $e_D \in E$ be the edge corresponding to interval D in TG connecting vertices $v_i, v_j \in V$. Then we say that the graph resulting from removing edge e_D and adding a new vertex $v = (\lambda, \sigma)$ plus edges (v, v_i) and (v, v_j) both labeled as e_D, is an extension of the transitions graph.

We define the extended transitions graph as the transitions graph with the extensions corresponding to the starting and ending solution instances of a reachability problem, if needed.

Figure 5.18 is the extended graph derived from the transitions graph in Figure 5.17 for a reachability problem with starting instance $I_s = T(5, \sigma_1)$, in interval A, and ending instance $I_e = T(5, \sigma_4)$, in interval G. Edge E_s is the one labeled with the interval's name A, and joins vertex (11.42, σ_1) with vertex (3, σ_1). Then, edge E_s is removed and vertex (5, σ_1) is created, as well as edges joining it to vertices (11.42, σ_1) and (3, σ_1). Analogously, edge E_e is removed and vertex (5, σ_4) is created. Edges joining it to (11.42, σ_4) and (3, σ_4) are also added.

In general, an edge path in a transitions graph that solves the reachability problem does not have to be unique. Among all the possible paths solving the reachability problem, we focus on those which are *optimal* in the sense of minimizing the arc length of the variant

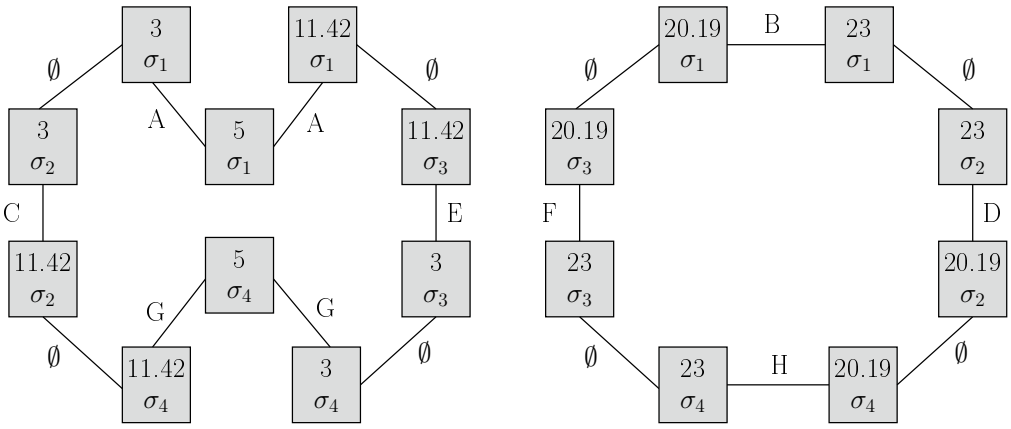


Figure 5.18: Extended transitions graph derived from the transitions graph in Figure 5.17 after adding the starting and ending vertices.

parameter.

Edges E_s and E_e are removed from the extended transitions graph because no optimal path between vertices V_s and V_e includes them. Consider that edge E_e joins vertices V_1, V_2, which are also joined to V_e, refer to Figure 5.19. Any optimal path arriving to V_1 or V_2 will visit directly V_e instead of the other vertex V_2 or V_1.

Among the techniques that have been developed to select an specific path in a weighted graph, if one exists, we have applied the A* algorithm, [91]. In our implementation, outlined in Algorithms 10 and 11, we consider that instances I are given as a pair (λ, σ), and the structure Edge stores two fields, source and sink, standing for the source and sink of the edge, respectively.

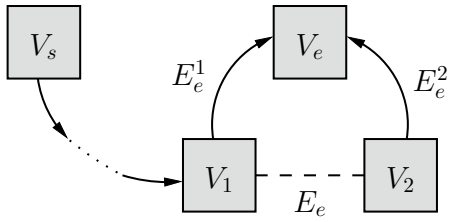


Figure 5.19: Schematic representation of a path solving the reachability problem from vertex V_s to vertex V_e. No optimal path between vertices V_s and V_e includes the dashed line.

Algorithm 10 Pathfinding

Input: TG, the transitions graph
I_s, starting instance
I_e, ending instance
Output: P, the path that leads from I_s to I_e, if one exists

Identify the domain interval D_s of I_s
Identify the domain interval D_e of I_e
if $D_s == D_e$ **then**
P = ((λ_s, σ_s), (λ_e, σ_e))
else
Compute the extended transitions graph ETG (TG, I_s, D_s, I_e, D_e)
P = A*(ETG, I_s, I_e)
end if

Algorithm 11 Compute ETG

Input: TG, the transitions graph
I_s, the starting instance
D_s, the domain interval of I_s
I_e, the ending instance
D_e, the domain interval of I_e
Output: ETG, the extended transitions graph

E_s = Edge labeled with D_s
E_e = Edge labeled with D_e
$V = V \cup \{(\lambda_s, \sigma_s), (\lambda_e, \sigma_e)\}$
$E = E \setminus \{ E_s, E_e \}$
$E = E \cup \{$ (E_s.source, (λ_s, σ_s)), (E_s.sink, (λ_s, σ_s)), (E_e.source, (λ_e, σ_e)), (E_e.sink, (λ_e, σ_e))$\}$

To compute the optimal path which solves the reachability problem from the starting vertex $V_s = (\lambda_s, \sigma_s)$ to the ending vertex $V_e = (\lambda_e, \sigma_e)$, we feed the A* algorithm with the extended transitions graph. The path-cost function used in A* is defined as

$$g(V) = \sum_{e \in P} w(e)$$

where e is an edge in P, a path from V_s to the current vertex $V = (\lambda, \sigma)$. Indeed, the path-cost function is equivalent to the sum of the weights of all the visited intervals edges, for the transitions edges have weight zero.

In the case that no improper transition is visited, two consecutive vertices in a path share either the parameter value or the index assignment. Consecutive vertices with the same index assignment are joined by an intervals edge, while consecutive vertices with the same parameter value are joined by a transitions edge. Thus, in graphs with no improper transitions, such as graphs related to distance type variant parameter problems, either edges have non-zero weight or the two parameter values are the same. Then, for distance type variant parameter problems, the following relation holds:

$$g(V) = |\lambda_1 - \lambda_s| + \sum_{i=1}^{n-1} |\lambda_{i+1} - \lambda_i| + |\lambda - \lambda_n|$$

where λ_i is the critical value in $V_i = (\lambda_i, \sigma_i)$, the i-th visited vertex of the path. Clearly, this relation is not true for graphs with improper transitions because the parameter values of the vertices related by those edges are different.

We define the heuristic to estimate the distance from a current vertex $V = (\lambda, \sigma)$ to the goal $V_e = (\lambda_e, \sigma_e)$ for distance-type parameters as the shortest arc of the variant parameter λ that must be traced to reach V_e, that is,

$$h(V) = |\lambda - \lambda_e|$$

Let us prove that $h(V) = |\lambda - \lambda_e|$ is admissible as required by the A* algorithm for distance variant parameters.

Theorem 5.2.3
For distance variant parameters, the heuristic $h(\lambda) = |\lambda - \lambda_e|$ to estimate the distance from the current vertex $V = (\lambda, \sigma)$ to the ending vertex $V_e = (\lambda_e, \sigma_e)$ does not overestimate the distance to the goal.

Proof
We consider an arbitrary vertex $V = (\lambda, \sigma)$ and compute the path P from V to the ending vertex $V_e = (\lambda_e, \sigma_e)$. Define the parameter value λ_i as the parameter value of the vertex

V_i, where V_i is the i-th visited vertex of the path P, and consider that P has n vertices plus V and V_e. Then, the distance to the goal is

$$w(P) = g(V_e) = \sum_{e \in P} w(e) = |\lambda_1 - \lambda| + \sum_{i=1}^{n-1} |\lambda_{i+1} - \lambda_i| + |\lambda_e - \lambda_n|$$

We prove that $w(P) \geq h(V)$. We apply successively the triangular inequality.

$$w(P) = |\lambda_1 - \lambda| + \sum_{i=1}^{n-1} |\lambda_{i+1} - \lambda_i| + |\lambda_e - \lambda_n| =$$

$$= |\lambda_1 - \lambda| + |\lambda_2 - \lambda_1| + \sum_{i=2}^{n-1} |\lambda_{i+1} - \lambda_i| + |\lambda_e - \lambda_n| \geq$$

$$\geq |\lambda_2 - \lambda| + \sum_{i=2}^{n-1} |\lambda_{i+1} - \lambda_i| + |\lambda_e - \lambda_n| \geq$$

$$\geq |\lambda_3 - \lambda| + \sum_{i=3}^{n-1} |\lambda_{i+1} - \lambda_i| + |\lambda_e - \lambda_n| \geq$$

$$\vdots$$

$$\geq |\lambda_n - \lambda| + |\lambda_e - \lambda_n| \geq$$

$$\geq |\lambda_e - \lambda| = h(V)$$

□

Now we consider angle variant parameters. Figure 5.20a shows a configuration for which $g(V)$ as defined above does overestimate the distance to the goal, if the parameter is an angle and there exists an improper transition between intervals A and B. In order to consider such configurations, we define a new heuristic as

$$h_\alpha(\lambda) = \min(|\lambda_e - \lambda|, |\lambda - \pi/2| + |\lambda_e + \pi/2|, |\lambda + \pi/2| + |\lambda_e - \pi/2|)$$

This heuristic considers the distances to the goal in the three cases depicted in Figure 5.20, and chooses the minimum one. In the following theorem, we prove that this heuristic is also admissible.

Theorem 5.2.4
For angle variant parameters, the heuristic $h_\alpha(\lambda) = \min(|\lambda - \lambda_e|, |\pi/2 - \lambda| + |\lambda_e + \pi/2|, |\pi/2 + \lambda| + |\lambda_e - \pi/2|)$ *to estimate the distance from the current vertex* $V = (\lambda, \sigma)$ *to the ending vertex* $V_e = (\lambda_e, \sigma_e)$ *does not overestimate the distance to the goal.*

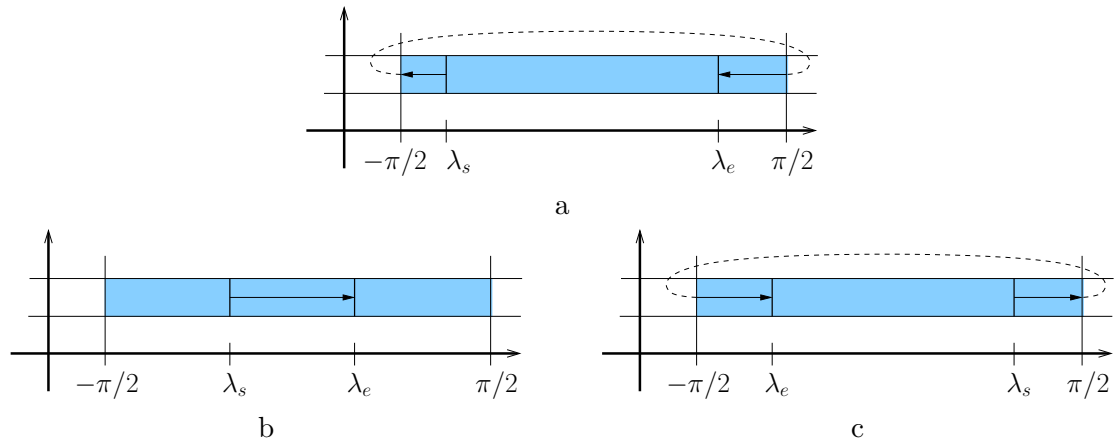


Figure 5.20: Angle variant parameters. Three possible configurations giving rise to three different values for the minimum distance covered by a path from the current vertex with parameter value λ to the final vertex with parameter value λ_e.

Proof

We consider an arbitrary vertex $V = (\lambda, \sigma)$ and compute the path P from V to the ending vertex $V_e = (\lambda_e, \sigma_e)$. Define the parameter value λ_i as the parameter value of the vertex V_i, where V_i is the i-eth visited vertex of the path P, and consider that P has n vertices apart from V and V_e. Then, the distance to the goal depends on the number of improper transitions the path visits. We prove that $w(P) \geq h_\alpha(V)$.

If the path visits no improper transitions edge, the distance to the goal can be computed applying Theorem 5.2.3, $w(P) \geq h(V) \geq h_\alpha(V)$. This is the case shown in Figure 5.20b.

If the path visits one improper transition, consider that the path visits exactly s vertices before, apart from V. The improper transition can be either from $-\pi/2$ to $\pi/2$, to which we refer as clockwise, or from $\pi/2$ to $-\pi/2$, to which we refer as counter-clockwise.

Consider first the case in which the improper transition is clockwise. Consider the simplest case in which the two intervals including the initial and final solution instances, with parameter values λ and λ_e respectively, are joined by the improper transition, Figure 5.20c. Then,

$$w(P) = |\pi/2 - \lambda| + |\lambda_e + \pi/2| \geq h_\alpha(V)$$

In the case that other intervals edges exist before or after the improper transition, triangular inequalities yield

$$w(P) = |\lambda_1 - \lambda| + |\pi/2 - \lambda_1| + |\lambda_e + \pi/2| \geq |\pi/2 - \lambda| + |\lambda_e + \pi/2| \geq h_\alpha(V)$$

Consider now the case in which the improper transition is counter-clockwise. Consider again the simplest case in which the two intervals including the initial and final solution instances, with parameter values λ and λ_e respectively, are joined by the improper transition, Figure 5.20a. Then,

$$w(P) = |-\pi/2 - \lambda| + |\lambda_e - \pi/2| \geq h_\alpha(V)$$

As above, ther intervals edges exist before or after the improper transition, triangular inequalities yield

$$w(P) = |\lambda_1 - \lambda| + |-\pi/2 - \lambda_1| + |\lambda_e - \pi/2| \geq |-\pi/2 - \lambda| + |\lambda_e - \pi/2| \geq h_\alpha(V)$$

If the path visits more than one improper transition, we can split the path P in different subpaths P_i, each of them including only one improper transition. Then, the weight of each path P_i is defined by the relations above. Clearly, $w(P) = \sum w(P_i)$.

Consider a path P split in two subpaths P_1, P_2, where P_1 begins at V and ends at $V_s = (\lambda_s, \sigma_s)$, and P_2 begins at V_s and ends at V_e. We prove that, if $w(P_1) \geq h_\alpha(V)$ and $w(P_2) \geq h_\alpha(V_s)$, then $w(P) \geq h_\alpha(V)$. We analyze three cases, depending on the kind of improper transition the subpaths include.

1. P_1 includes a clockwise improper transition, P_2 includes a clockwise improper transition. Then,

$$w(P) = w(P_1) + w(P_2) = |-\pi/2 - \lambda| + |\lambda_s - \pi/2| + |-\pi/2 - \lambda_s| + |\lambda_e - \pi/2| \geq$$

$$\geq |\lambda_e - \lambda_s| + |\lambda - \lambda_s| \geq |\lambda_e - \lambda| \geq h_\alpha(V)$$

2. P_1 includes a clockwise improper transition, P_2 includes a counter-clockwise improper transition. Then,

$$w(P) = w(P_1) + w(P_2) = |-\pi/2 - \lambda| + |\lambda_s - \pi/2| + |\pi/2 - \lambda_s| + |\lambda_e + \pi/2| \geq$$

$$\geq |\lambda_e - \lambda| + 2|\lambda_s - \pi/2| \geq |\lambda_e - \lambda| \geq h_\alpha(V)$$

3. P_1 includes a counter-clockwise improper transition, P_2 includes a counter-clockwise improper transition. Then,

$$w(P) = w(P_1) + w(P_2) = |\pi/2 - \lambda| + |\lambda_s + \pi/2| + |\pi/2 - \lambda_s| + |\lambda_e + \pi/2| \geq$$

$$\geq |\lambda_e - \lambda| + 2|\lambda_s + \pi/2| \geq |\lambda_e - \lambda| \geq h_\alpha(V)$$

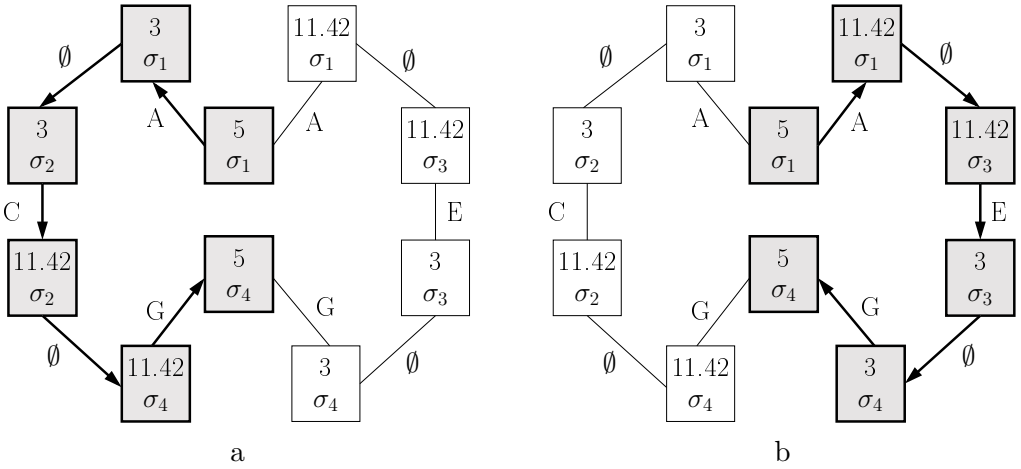


Figure 5.21: Two minimum paths output by Algorithm 10 that solve the reachability problem in Figure 5.16. $I_s = T(5, \sigma_1)$ and $I_e = T(5, \sigma_4)$. Grey vertices represent the path, and white vertices are the not visited vertices.

The fact that this methodology can be successively applied to paths with a finite number of improper transitions concludes the proof.

□

Figure 5.21 shows two paths with minimum variant parameter arc length computed by Algorithm 10 that solves the reachability problem whose domain is given in Figure 5.16. The starting instance $I_s \in A$ is defined by $\lambda_s = 5$ and $\sigma = \{+1, +1\}$, and the ending instance $I_e \in G$ is defined by $\lambda_e = 5$ and $\sigma = \{-1, -1\}$. The total variant parameter arc length for these paths is 16.84 and both paths are optimal.

5.3 Implementation and results

Our approach to solve the reachability problem has been implemented in the framework of the dynamic geometry system based on constructive geometric constraint solving described by Freixas *et al.* in [25]. The system has two components. One includes a user graphic interface and a constructive geometric constraint solver in charge of both defining the parametric geometric object and generating a construction plan that solves it. The other component, that we call the *dynamic selector*, defines the dynamic behavior of the geometric object and solves the reachability problem.

To illustrate how the implementation works, we will show a complete case study in which

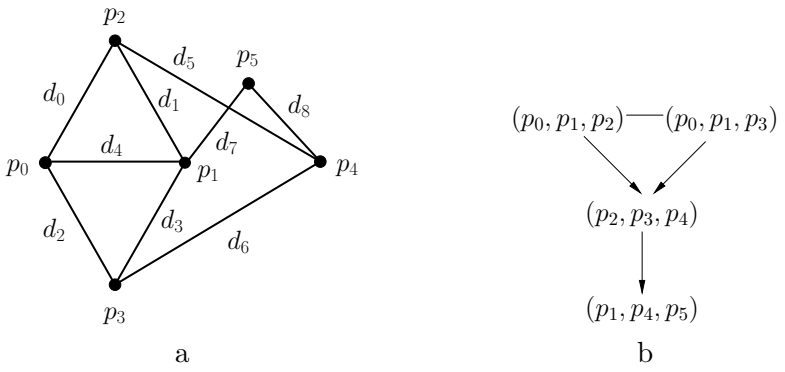


Figure 5.22: Geometric constraint problem with six points and nine point-point distances. a) Graph G. b) h-graph $\mathcal{H}(G)$ associated to G.

the system solves a reachability problem associated to the geometric constraint problem depicted in Figure 5.22a. This problem includes six points and nine point-point distances, where d_4 is the variant parameter and the distance constraint are assigned values

$$\begin{array}{cccc} d_0 = 2 & d_1 = 2 & d_2 = 1 & d_3 = 1 \\ d_5 = 0.7 & d_6 = 0.7 & d_7 = 0.8 & d_8 = 0.89 \end{array}$$

The h-graph associated to the problem is shown in Figure 5.22b. Clearly, the problem is composed by 4 tree-decomposition steps, (p_0, p_1, p_2), (p_0, p_1, p_3), (p_2, p_3, p_4) and (p_1, p_4, p_5). Problems (p_0, p_1, p_2), (p_0, p_1, p_3) depend directly on the variant parameter and problems (p_2, p_3, p_4), (p_1, p_4, p_5) depend indirectly on it.

Once the dynamic problem has been introduced in the system through the user graphic interface, the constructive geometric constraint solver computes the construction plan that solves the underlaying geometric constraint problem shown in Figure 5.23. For more details on this construction, see [25].

Now, the dynamic selector will define and solve the reachability problem. We have divided the dynamic selector into two different parts. The part in charge of defining the reachability problem is called *endpoints selector*, and the part in charge of solving it is called *reachability simulator*.

The main task of the endpoints selector consists in computing the domain of the variant parameter and the transitions graph of the problem, using the algorithms explained in Section 4.5, Chapter 4. Once the domain and the tree-decomposition are computed, the selector suggests to the user arbitrary initial and final instances belonging to a connected component of the transitions graph.

1.	p_0	$= origin()$	8.	p_3	$= intCC(c_2, c_3, s_1)$
2.	p_1	$= distD(p_0, d_4)$	9.	c_4	$= circleCR(p_2, d_5)$
3.	c_0	$= circleCR(p_0, d_0)$	10.	c_5	$= circleCR(p_3, d_6)$
4.	c_1	$= circleCR(p_1, d_1)$	11.	p_4	$= intCC(c_4, c_5, s_2)$
5.	p_2	$= intCC(c_0, c_1, s_0)$	12.	c_6	$= circleCR(p_0, d_7)$
6.	c_2	$= circleCR(p_0, d_2)$	13.	c_7	$= circleCR(p_4, d_8)$
7.	c_3	$= circleCR(p_1, d_3)$	14.	p_5	$= intCC(c_6, c_7, s_3)$

Figure 5.23: Construction plan given by the constructive geometric constraint solver for problem in Figure 5.22.

In our implementation, the system labels the intervals with consecutive integers. For a better understanding we have labeled them in this work with capital letters, but from now on we will label intervals with integers.

The reachability problem we solve is defined by the starting instance $I_s = T(0.5, \sigma_4)$ and the ending instance $I_e = T(0.5, \sigma_2)$. Figure 5.24 shows the domain of the variant parameter in this problem together with the set of continuous transitions among the intervals. The transitions graph of this problem, although never displayed by the system, is shown in Figure 5.25.

Finally, the reachability simulator figures out the extended transitions graph for the starting and ending instances selected by the user, shown in Figure 5.26. The search for

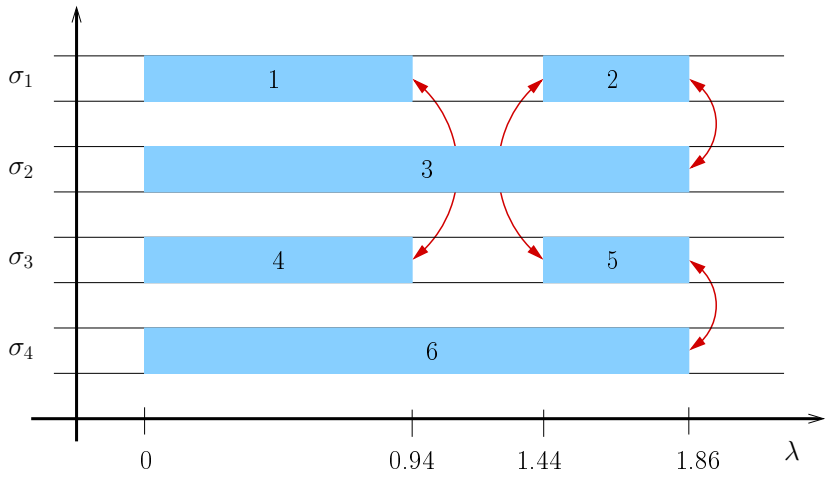


Figure 5.24: Domain of the variant parameter of the problem in Figure 5.22. The set of continuous transitions betweene intervals are displayed as arrows.

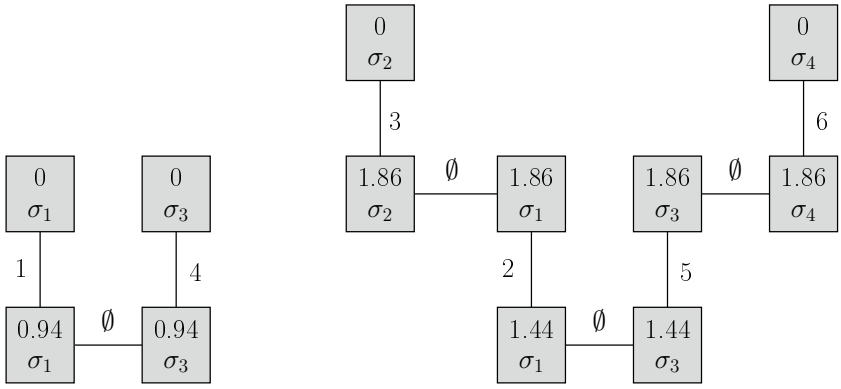


Figure 5.25: Transitions graph of the problem in Figure 5.22.

the minimum path is performed according to Section 5.2.2, giving rise to the path shown in Figure 5.27.

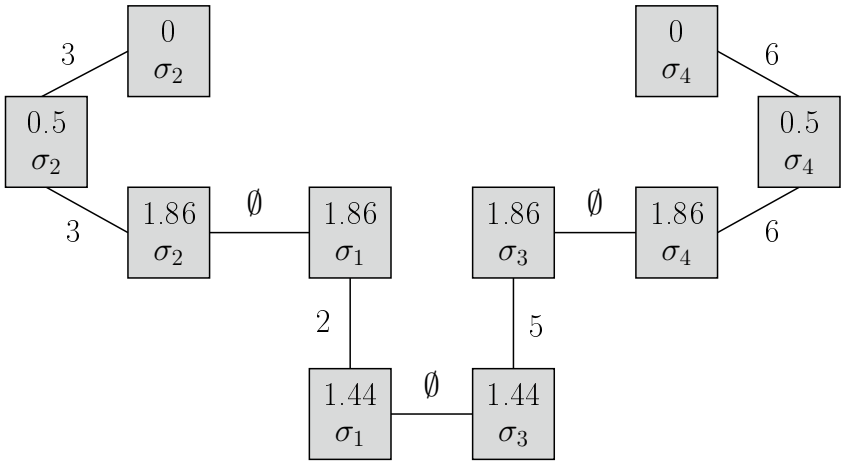


Figure 5.26: Extended transitions graph for the reachability problem with initial instance $I_s = T(0.5, \sigma_4)$ and ending instance $I_e = T(0.5, \sigma_2)$.

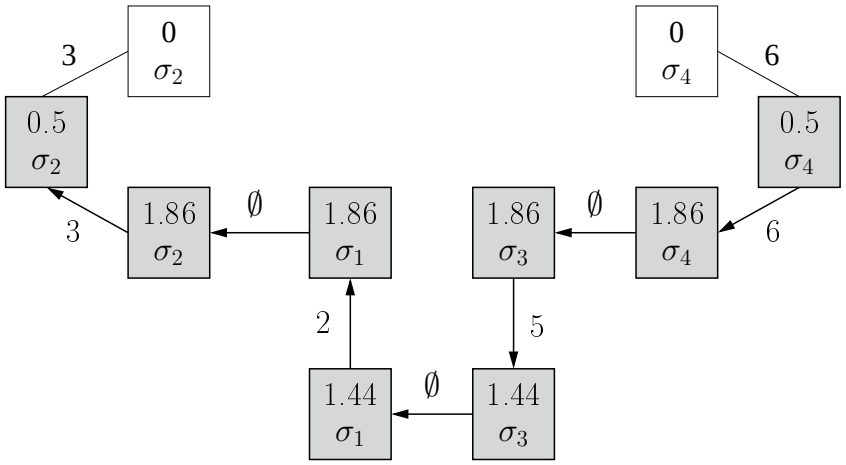


Figure 5.27: Minimum path computed by the system for the reachability problem with initial instance $I_s = T(0.5, \sigma_4)$ and ending instance $I_e = T(0.5, \sigma_2)$.

5.4 Conclusions

In this chapter we have presented a solution to the reachability problem in geometric constraint dynamic geometry. The technique assumes the existence of a construction plan for the geometric constraint problem under study and is based on the analysis of the problem's domain and the continuous transitions among its domain intervals. Continuous transitions allow to perform changes in the index assignments of a dynamic evaluation in such a way that the resulting evaluation is continuous.

A graph capturing all the continuous transitions arising in the domain of the problem, called transitions graph, is defined. Using the A* algorithm in the transitions graph the method finds a path, if possible, which solves the reachability problem.

The technique has been implemented on top of the dynamic geometry system based on constructive geometric constraint solving presented by Freixas *et al.* in [23], and it has proven to be both effective and efficient from a practical point of view. The execution time is around 40 milliseconds for the computation of the domain in the case entailing six points and nine distances proposed in Section 5.3, and 1 millisecond for the minimum path.

CHAPTER 6

The tracing problem

You have brains in your head. You have feet in your shoes.
You can steer yourself any direction you choose.
You're on your own. And you know what you know.
And YOU are the one who'll decide where to go...

Dr. Seuss, Oh, the Places You'll Go!

When changing variable values or freely moving different elements of a geometric construction on a dynamic geometry environment, the user expects to see the changes of his interaction immediately reflected in the construction. However, often the requested geometric construction is not unique under the established conditions. The system must decide, therefore, for each time instant, which one of the different possible instances of the construction the user is expecting to see. To display the correct solution instance at every time instance is known as the *tracing problem*. In this chapter we give a more formal definition of this problem as well as our approach to the solution.

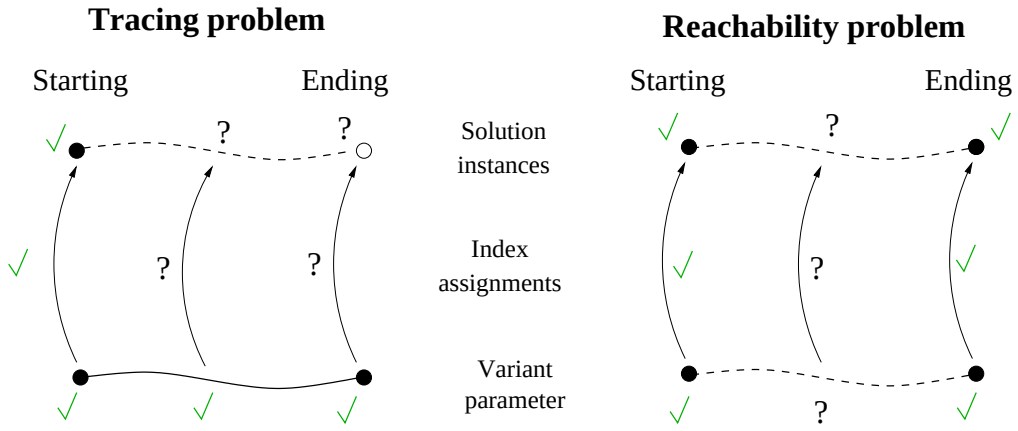


Figure 6.1: Scheme of the given (check marks) and on demand (question marks) information in the tracing and the reachability problems. From Denner-Broser, [16].

6.1 Definition of the tracing problem

The tracing problem is one of the most challenging problems in dynamic geometry. In simple words, given an initial instance of the problem and the path followed by the variant parameter, the tracing problem consists of showing, at every moment, a solution instance from the set of possible instances of the system. This includes solving the root identification problem as well as the handling of the points where no possible solution exists. The aim of the tracing problem is to show the solution instance the user is expecting to see, which is called the *intended solution*.

Despite the resemblances between the tracing and the reachability problems, there are two main features that distinguish them. Following Denner-Broser, [16], we outline both problems with the scheme shown in Figure 6.1. Given inputs of information are displayed by a check mark. Information in demand for each problem is displayed by a question mark.

Assuming that the solution instance at every instant is defined once the variant parameter and the index assignment are known, the first row in Figure 6.1 gives no further information than the already given by the second and the third ones. It is shown that the tracing and the reachability problem share three input information: the variant parameter of the starting and ending instances and the index assignment of the starting one.

The reachability problem also has as input information the index assignment related to the ending instance, which means that the ending instance is known. The problem consists on finding a variant parameter path and the associated index assignment allowing a dynamic evaluation from the starting instance to the ending one, as seen in Chapter 5.

On the contrary, the tracing problem has as extra input information the path followed by the variant parameter from the initial instance to the ending one, $\lambda(t)$. No information about the index assignment at any point different to the starting one is given. The problem consists on choosing the right instance among those possible instances at each path point.

Many formal definitions of the tracing problem have been stated in the literature, see for example [15, 17, 18, 70, 83]. We formally state the tracing problem as follows.

> Let $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ be a geometric constraint problem with one variant parameter λ and let $T(\sigma, \lambda)$ be a construction plan that solves Π. Let $I_s = T(\lambda_s, \sigma_s)$ be a starting instance of T, and $\lambda(t) : [0, 1] \to \mathbb{R}^+$ a continuous path of the variant parameter. Decide the assignments of index
>
> $$\sigma(t) : [0, 1] \to \{-1, +1\}^n$$
>
> for which there is a dynamic evaluation $T(\sigma, \lambda)(t)$ feasible for every $t \in [0, 1]$ such that $T(\sigma, \lambda)(0) = I_s$.

Notice that no continuity requirements have been specified in the definition of the problem. If we want the dynamic evaluation to be continuous, we must define a new problem, which we call the *continuous tracing problem*. This variant of the tracing problem is formally stated as follows.

> Let $\Pi =< \Pi_G, \Pi_C, \Pi_P >$ be a geometric constraint problem with one variant parameter λ and let $T(\sigma, \lambda)$ be a construction plan that solves Π. Let $I_s = T(\lambda_s, \sigma_s)$ be a starting instance of T, and $\lambda(t) : [0, 1] \to \mathbb{R}^+$ a continuous path of the variant parameter. Decide the assignments of index
>
> $$\sigma(t) : [0, 1] \to \{-1, +1\}^n$$
>
> for which there is a continuous evaluation $T(\sigma, \lambda)(t)$ feasible for every $t \in [0, 1]$ such that $T(\sigma, \lambda)(0) = I_s$.

In this new statement of the problem, the index assignment $\sigma(t)$ must ensure the continuity of the dynamic evaluation $T(\sigma, \lambda)(t)$.

Users are used to a world where objects do not suffer from jumps in their behaviors. Thus, the natural solution to the tracing problem should show a continuous movement of all the involved elements. The natural tracing problem is then the continuous one.

6.2 Solution to the tracing problem

In this section we recall some of the approaches in the literature to solve the tracing problem, and present our own approach. We discuss the differences between the methods and analyze their characteristics. Some properties of our method are highlighted at the end of the section.

6.2.1 Previous approaches

The tracing problem has been addressed for a long time in the literature. In Richter-Gebert [90], it has been shown to be a NP-hard problem by giving a reduction to the 3-SAT problem, one of the standard NP-complete decision problems. Also the complexity of other related tracing problems has been established in [83, 90].

Different approaches to the solution of this problem have been reported, but they are all based on the same idea: to prevent the construction of the solution at critical values, inherent to the problem at hand. Following this idea we find the work of Kortenkamp, [70], which suggests a proximity criterion which chooses as next instance the nearest solution to the actual one. The method, as expected, gives accurate results as long as the path does not contain critical values. There, the system has no method to distinguish between different solutions. As strategy to avoid the undesired critical values, Kortenkamp proposes a detour in the path of the variant parameter through the complex plane.

Denner-Broser in [16, 17, 18], develops a full theory about detours and alternative paths through the complex plane with an algorithm which proceeds stepwise and detects potential critical values in advance using interval arithmetic. After detecting a critical value, the singularity is avoided by a detour in $\mathbb{C}$. Denner-Broser also suggests in [15] an approach based on avoiding the critical values by computing a Voronoi diagram where the sites are the critical values. The solution to the tracing problem is given by the edges of the diagram.

Our approach, however, differs substantially from these approaches. We present a method in which critical values are not to be avoided, but used as the place where continuous transitions are defined to connect different intervals of the domain to allow continuous dynamic evaluations.

6.2.2 An approach to the solution of the tracing problem

We present in this section an approach to solve the tracing problem, both the simple and the continuous one, in the framework of the dynamic geometry system based on constructive

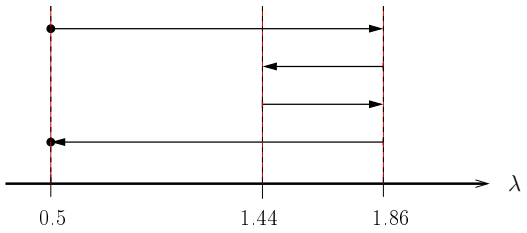


Figure 6.2: Definition of the tracing problem by means of a scheme

geometric constraint solving described by Freixas *et al.* in [24]. In this framework, after the analysis and computation of the domain, critical values and continuous transitions are identified, leading to a map of the domain which allows the user to plan which one of the possible paths the tracing is going to take. When the variant parameter path reaches a critical value with multiple solution instances, the system is able to distinguish the different solutions which converge at that point, and identifies the transitions leading to a continuous behavior of the construction.

Nevertheless, the solution to the continuous tracing problem is not necessarily unique. As long as the variant parameter path goes over a point of continuous transition, a possible change of the index assignment can be done in such a way that the corresponding dynamic evaluation is continuous, leading possibly to many different dynamic evaluations which solve the tracing problem at hand.

Assuming that two different continuous evaluations can be established fulfilling the conditions to solve a given tracing problem, since the intended solution is the one the user is "expecting to see", only he is able to decide which one is the correct one. We give then this responsibility to the user. In our system, the user has all the information about the possible paths and transitions, and he can choose the intended solution he is expecting to see.

As an example of tracing problem with more than one continuous solution we propose to solve the tracing problem defined by the path solution to the reachability problem stated in Chapter 5, Section 5.3. The path followed by the variant parameter is schematically represented in Figure 6.2. From top to bottom and from left to right, this figure represents how the variant parameter increases from 0.5 to 1.86, then decreases until 1.44, increases again up to 1.86 and finally decreases to reach the final value 0.5.

Figure 6.3 shows the continuous evaluation output by the dynamic selector, which also solves the tracing problem at hand. An index assignment has been chosen for every point in the path $\lambda(t)$, changing the sign exactly at the transition points and assuring the final continuity of the dynamic evaluation, which traverses intervals 6, 5, 2 and 3. The ending

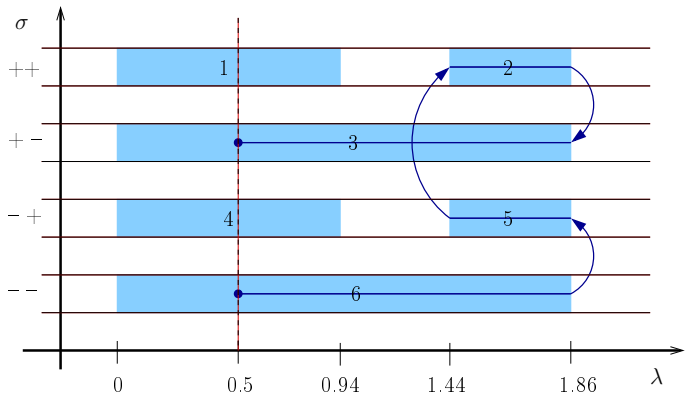


Figure 6.3: Solution to the tracing problem corresponding to the solution to the reachability problem in Chapter 5, Section 5.3

instance is $T(\sigma_2, 0.5)$.

Nevertheless, other continuous evaluations present also solutions to the considered tracing problem. Figure 6.4 represents a dynamic evaluation assigning to each point in the path $\lambda(t)$ an index assignment which also assures the continuity. The continuous evaluation traverses intervals 6 and 5, where no change of index occurs. The ending instance is the same as the starting one, $T(\sigma_4, 0.5)$.

Figure 6.5 represents another dynamic evaluation which also solves the tracing problem at hand. In this case, no change of index assignment is considered, and each variant parameter in the path $\lambda(t)$ we associate the same index assignment, σ_4. The final instance is again the same as the starting one, $T(\sigma_4, 0.5)$. Notice that no criterion has been established however to allow the system to select one of them as the intended solution. Only the user can answer this question.

6.2.3 On continuity and determinism

As seen in Section 2.4, continuity and determinism are two of the most important and desirable characteristics of dynamic geometry systems. As seen there, continuity assures the non-existence of undesirable or unexpected *jumps* while geometric objects move on the screen, avoiding the existence of erratic behaviour in the user interface. The importance of continuity derives from the fact that real systems are continuous, and so must be their representations. Determinism contributes to system stability by guaranting the same behaviour once the initial conditions have been fixed.

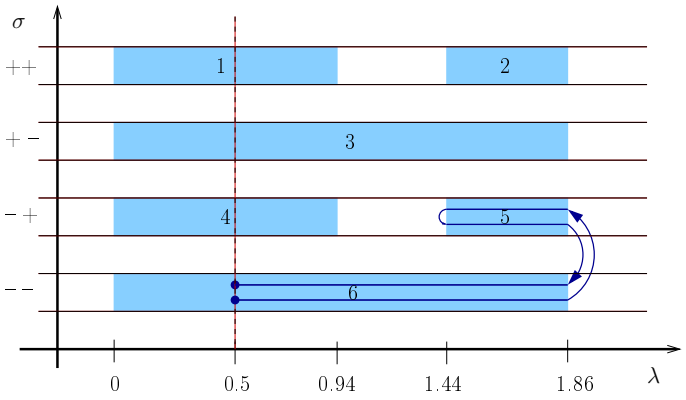


Figure 6.4: Another solution to the tracing problem traversing only two intervals.

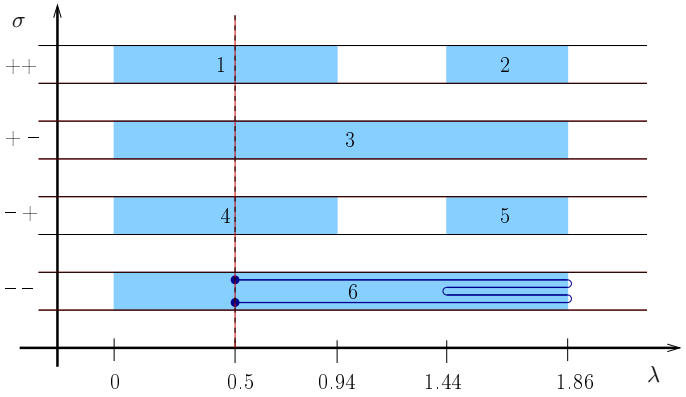
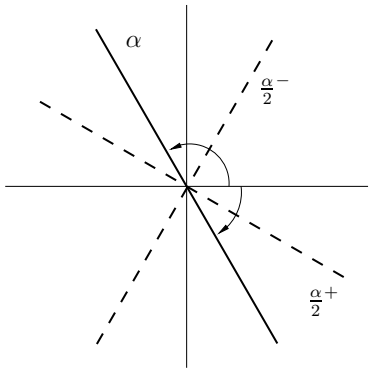


Figure 6.5: Other solution to the tracing problem which associates to every point in the variant parameter path the same index assignment.



a

	α	$\alpha/2$	$\alpha/4$
σ_1	120°	60°	30°
σ_2	300°	150°	75°
σ_3	480°	240°	120°
σ_4	660°	330°	165°

b

Figure 6.6: Bisector and double bisector. a) Possible bisectors of the angle α. b) Values of the angle, bisector and double bisector for the four different assignments of the problem with two bisectors.

It has been claimed that continuity and determinism are mutually exclusive when bisectors of angles are included in the set of basic operations the system handles, [16, 70, 83]. We shall see that this is not the case in our system.

Notice that, by definition, when increasing an angle variant parameter with respect to a fixed point, the bisector of that angle covers half the angular distance of the angle itself. Therefore, the bisector angular speed is half the angular speed of the angle. By the same principle, the bisector of the bisector af an angle covers a fourth part of the angular distance covered by the original angle.

If we allow the angle variant parameter to increase from 0 up to 2π, the starting and ending values of that angle are coincident modulo 2π. However, the bisector angle will be at π and the double bisector angle will be at $\pi/2$. In this situation, according to the definitions of Kortenkamp, [70], continuity and determinism are considered as mutually exclusive.

Nevertheless, our approach will have determinism and continuity simultaneously even in this case. A sign will be defined considering the two possible cases for the angle, leading to the two possible bisectors depicted in Figure 6.6a. Thus, the bisector in our system will be defined as:

> Let angle $\alpha \in [-\pi/2, \pi/2]$. With the bisector of α we associate a sign, say σ_0. The bisector is defined as $\alpha/2 \in [-\pi/2, \pi/2]$ for $\sigma_0 = +1$ and $\alpha/2 + \pi/2 \mod \pi \in [-\pi/2, \pi/2]$ for $\sigma_0 = -1$.

The application of this definition to the problem with two bisectors will produce an

index with two signs, one for each bisector included in the construction, leading to four different possible index assignments. The values of the angle α, the bisector and the double bisector for the four different assignments are depicted in Table 6.6b. Figure 6.7 shows from left to right and from top to bottom the four different instances of the problem related to the index assignments $\sigma_1, \sigma_2, \sigma_3$ and σ_4, respectively. The angle α is limited by a thick line, the bisector by a dashed line and the double bisector by a dotted line. Arrows point to the value of the angles inside $[-\pi/2, \pi/2]$, that is, modulo π.

Consider now the top left figure in Figure 6.7 representing the configuration corresponding to the index assignment σ_1. If we increase α in 180° through the X axis, the bisector $\alpha/2$ traverses the Y axis and the resulting configuration is the one depicted in the top right figure, associated to the index assignment σ_2. Increasing in other 180° the value of α, the dashed line traverses the X axis while the dotted line traverses the Y axis. The resulting situation is depicted in the bottom left figure, related to the index assignment σ_3. Another increment of 180° in α makes the dashed line traverse the Y axis again, adopting the configuration represented in the bottom right figure and associated to the index assignment σ_4. Notice that a last increment of 180° will make both the dashed and the dotted line traverse the X axis, arriving to the initial configuration in the top left figure corresponding to the index assignment σ_1.

The domain of the problem defined by an angle α, the bisector $\alpha/2$ and the bisector of the bisector $\alpha/4$ is shown in Figure 6.8. Notice that the continuous transitions between intervals reflect the behavior explained above. Each index assignment σ_i is connected by an improper transition to the index assignment σ_{i+1} except σ_4, connected to σ_1. This domain and the included improper transitions capture the original behavior of the problem, which allowed two complete turns of the angle variant parameter before the current instance and the initial one became coincident. That feature gives to our approach soundness and robustness.

6.3 Implementation

The implementation of our approach to solve the tracing problem is build on top of the dynamic geometry system based on constructive geometric constraint solving described by Freixas *et al.* in [24], on top of which our approach to the reachability problem was also implemented, Chapter 5, Section 5.3.

The specific functional unit in charge of actually solving the tracing problem is the *simulator*. Once the user has defined the specific dynamic evaluation he is expecting to see, the simulator is in charge of showing it in the screen.

The simulator simply receives the selected path in the transitions graph, which has

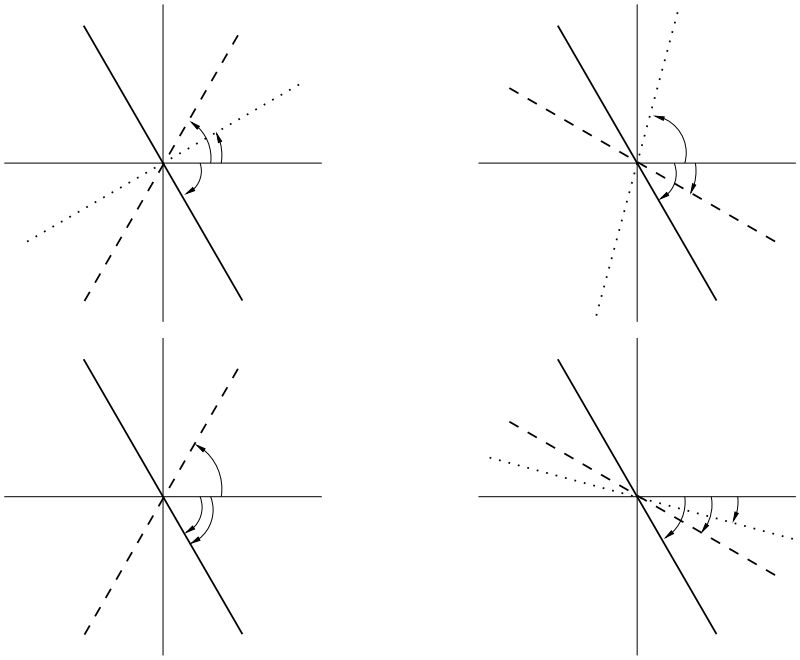


Figure 6.7: Four possible configurations for the double bisector, allowing determinism and continuity. From left to right and from top to bottom, solution instances corresponding to index assignments $\sigma_1, \sigma_2, \sigma_3$ and σ_4 are shown.

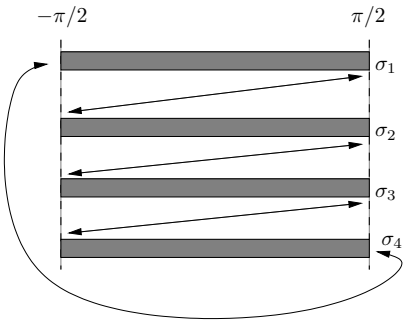


Figure 6.8: The domain of the hypothetic problem with two bisectors.

been extended if needed with the starting and ending instances the user is expecting to see, and shows the resulting dynamic evaluation of the problem in the *simulator window*. The visualization of all the solution instances of a dynamic evaluation in the screen is called a *simulation*.

Figure 6.9 shows a simulator window which has six panels divided into three main areas. The top area is composed of two panels, which, from left to right, are called the *display panel* and the *domain panel*. The display panel is the panel where the current instance is show. Initially it displays the starting instance of the path the user has selected. The simulation, if possible, will be displayed in this window.

In the domain panel, the domain of the variant parameter for the geometric problem at hand is shown. It also gives information about the current instance by displaying a red vertical line indicating the current value of the variant parameter, and by displaying in a different color the current interval.

The middle area is made of three different panels, reporting information related to the problem at hand. They are called the *information panels*. The left information panel displays information about the geometric constraint problem. The right upper information panel shows the index of the current instance. The right lower information panel shows the different files used in the process.

The bottom area includes only the *interaction panel*, where some buttons lie. Button *Go* starts the simulation, button *Stop* stops it and the lower scroll adjusts its velocity. In this panel the user can also define the initial and final instances of a reachability problem in the corresponding cells. In this case, button *Set* sets as current instance the initial instance of the new reachability problem, and proceeds to solve the reachability problem as described in Chapter 5, Section 5.3. The solution path is then stored as the new path of the tracing problem to show, and button *Go* starts again the simulation.

Figure 6.10 shows different instances of the simulation for the path in Figure 5.27, which solves the reachability problem with initial instance $I_s = T(0.5, \sigma_4)$ and ending instance $I_e = T(0.5, \sigma_2)$ described in Chapter 5, Section 5.3.

6.4 Conclusions

In this chapter we have presented an approach to solve the tracing problem in geometric constraint dynamic geometry. As in the case of the reachability problem, the method is based on the analysis of the problem's domain and the continuous transitions among its domain intervals. In this method, the user is asked to define the concrete behavior he is expecting to see, that is, the intended solution.

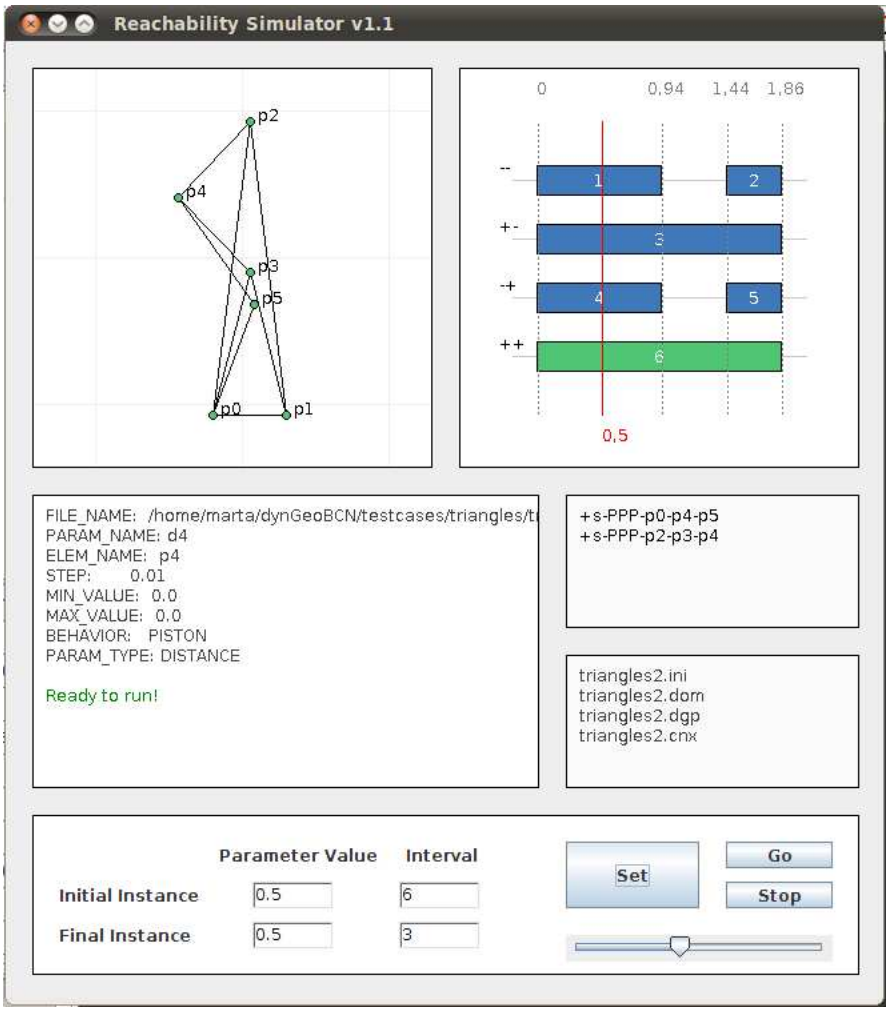


Figure 6.9: The reachability simulator window at the initial instance of the simulation.

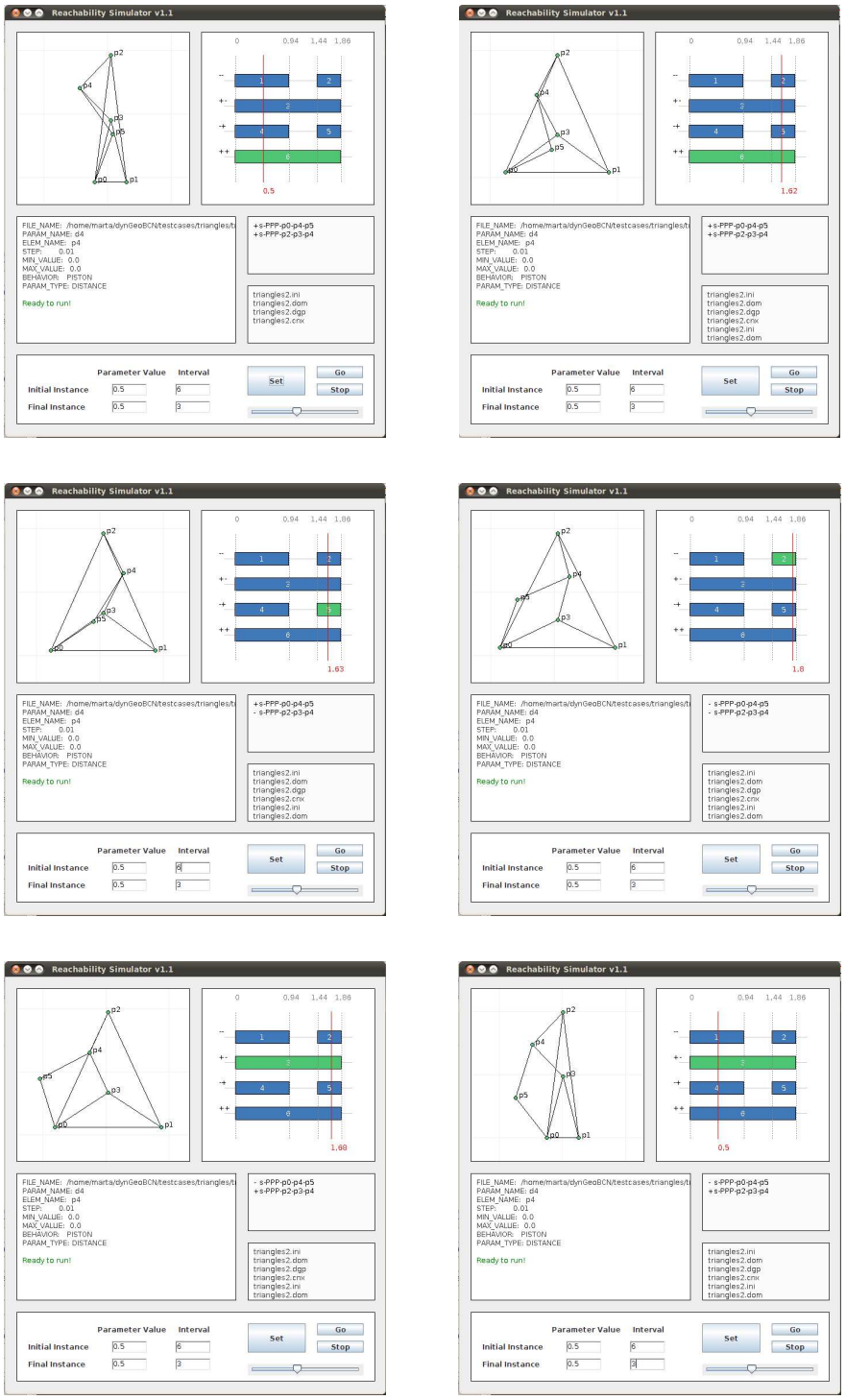


Figure 6.10: From left to right and from top to bottom, different instances in the tracing path for the tracing problem considered. The upper left image corresponds to the initial instance and the lower right image to the final one.

The technique has been also implemented on top of the dynamic geometry system based on constructive geometric constraint solving presented by Freixas *et al.* in [23] and allows continuity and determinism at the same time.

CHAPTER 7

Henneberg graphs and tree-decomposability

I personally believe we were put here
to build and not to destroy.

Red Skelton

Well-constrained graphs, also known as minimally rigid or Laman graphs, can be constructed by means of a sequence of two different construction steps called Henneberg steps. It is also known that not all the Laman graphs are tree-decomposable. In this chapter we study Henneberg steps and analyze thoroughly the relationship between the Laman graphs generated by such steps and tree-decomposability. We create an algorithm which generates tree-decomposable Laman graphs with a given order using Henneberg steps. The results reported in this chapter establish for the first time, as far as we know, some relationships between graph tree-decomposability and Henneberg graph constructions.

The chapter is divided into three main parts. In the first one, Henneberg steps and families are defined, and a characterization for the set of tree-decomposable graphs is given. The main aim of the first part is to determine the inclusion relations arising between Henneberg families and the set of tree-decomposable graphs. The second part of the chapter is devoted to establish the theoretical requirements necessary to assure the tree-decomposability of a tree-decomposable graph after the application of a Henneberg step. In the third part we present an authomatic process to generate tree-decomposable Laman

graphs with a given order by means of Henneberg steps and using h-graphs, introduced in Chapter 3.

7.1 Henneberg families and tree-decomposable graphs

Ernst Lebrecht Hennberg presented in 1911 a method to construct Laman graphs based on the composition of two construction steps which we usually know as Henneberg steps, [35]. The two different steps give rise to two families of graphs, constructed only with each one of the Henneberg steps. We devote this section to establish the inclusion relations arising between each of the so defined families and the set of tree-decomposable Laman graphs.

7.1.1 Henneberg steps and Henneberg families

Laman graphs have some elegant and useful features. One of the most important is that they can be constructed inductively from the triangle graph K_3 by means of the so called Henneberg steps [35], as seen for example in [94, 100]. We devote this section to introduce Henneberg steps.

In particular, there are two kinds of Henneberg steps.

1. *Henneberg I step* or simply HS1: Let $G = (V, E)$ be a graph and $v_1, v_2 \in V$ with $v_1 \neq v_2$. Consider a new vertex $v \notin V$. Then the graph $G^* = (V^*, E^*)$ with $V^* = V \cup \{v\}$ and $E^* = E \cup \{(v_1, v), (v_2, v)\}$ is the graph derived from G by a Henneberg I step. See Figure 7.1. We will say that such step involves vertices v_1, v_2.

2. *Henneberg II step*, or simply HS2: Let $G = (V, E)$ be a graph with $e = (v_1, v_2) \in E$ and $v_3 \in V$. Consider a new vertex $v \notin V$. Then the graph $G^* = (V^*, E^*)$ with $V^* = V \cup \{v\}$ and $E^* = E \cup \{(v_1, v), (v_2, v), (v_3, v)\} \backslash \{e\}$ is the graph derived from G by a Henneberg II step. See Figure 7.2. We will say that such step involves edge e and vertex v_3.

Notice that the application of a HS1 can be seen as the addition to a graph of a new degree 2 vertex v joined to the vertices v_1 and v_2. Besides, the application of a HS2 can be seen as the substitution in a graph of an edge $e = (v_1, v_2)$ by a new vertex v joined to the vertices of v_1, v_2 and to a different vertex v_3.

Definition 7.1.1
The sequence of Henneberg steps transforming the triangle graph K_3 into the graph G is called the Henneberg sequence of G.

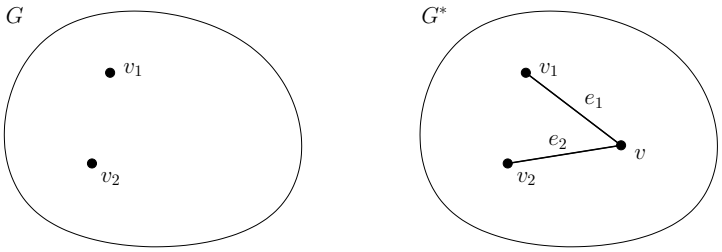


Figure 7.1: Henneberg I step. a) Graph G. b) Graph G^* derived from graph G by the application of a HS1.

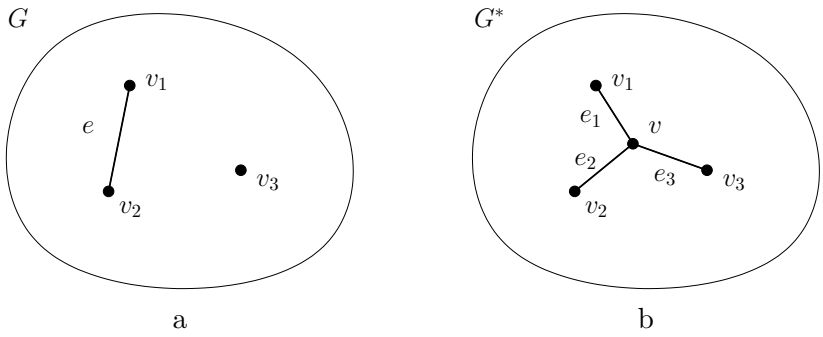


Figure 7.2: Henneberg II step. a) Graph G. b) Graph G^* derived from graph G by the application of a HS2.

Henneberg steps give rise to two different families of graphs, as defined in [7, 22]. A graph G belongs to the Henneberg I family, denoted H_I, if it can be constructed using only Henneberg steps of type I. The graph G belongs to the Henneberg II family, denoted H_{II}, if both Henneberg steps of type I and II are needed to build G. As shown in [35], H_{II} and the the set of Laman graphs are coincident.

The application of Henneberg steps can also be abstracted as a rewriting system where objects are graphs and the Henneberg steps are the rewriting rules. In this context, we will denote the application of a HS1 to the graph G as $G \Rightarrow_1 G^*$ and the application of a HS2 to G as $G \Rightarrow_2 G^*$. The application of an arbitrary Henneberg sequence to K_3 leading to G shall be denoted by $K_3 \Rightarrow_* G$. In order to identify two different Henneberg sequences, we shall use a supra index. The application of a Henneberg sequence including only Henneberg steps of type I to K_3 leading to G shall be denoted by $K_3 \Rightarrow_{1*} G$.

7.1.2 A characterization of tree-decomposable Laman graphs

In this section we propose a diferent point of view on tree-decomposable Laman graphs. We present a characterization of the set of tree-decomposable Laman graphs based on an operation called merging. We will see that the set of graphs constructed by means of this operation is exactly the set of tree-decomposable Laman graphs. A graph constructed by means of a sequence of merging operations is tree-decomposable, and the merging sequence can be understood as the composition of the decomopsition steps of the tree-decomposition in bottom-up direction.

There are different methods to generate tree-decomposable Laman graphs. Vila, in [106], characterizes the set of tree-decomposable Laman graphs as the set $\mathcal{L}$ generated from the triangle graph K_3 by the operation called *amalgamation*. We propose a similar characterization based on an operation called *merging*, which we define as follows. Refer to Figure 7.3.

Definition 7.1.2
Let $A = (V_A, E_A), B = (V_B, E_B), C = (V_C, E_C)$ be three graphs with $a_1, b_1 \in V_A$, $c_1, a_2 \in V_B$ and $b_2, c_2 \in V_C$. The graph $D = (V_D, E_D)$ is called the merging of graphs A, B and C if $V_D = V_A \cup V_B \cup V_C$ and $E_D = E_A \cup E_B \cup E_C$ after the identification of vertices $a_1 = a_2 = a$, $b_1 = b_2 = b$ and $c_1 = c_2 = c$.

Graphs A, B, C are called the clusters of the merging, and the identified vertices a, b, c, the merging vertices. The set of the three merging vertices of a merging is called the merging triple.

We define the set $\mathcal{T}$ as the set defined constructively by the following rules:

Definition 7.1.3

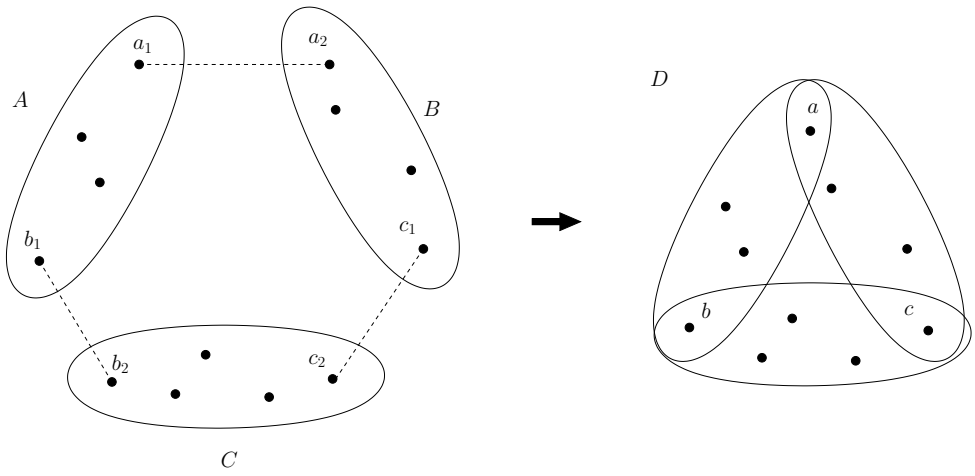


Figure 7.3: Merging of graphs A, B, C giving rise to graph D, with hinges a, b, c.

Consider the set of graphs $\mathcal{T}$ such that

1. *The triangle graph K_3 belongs to $\mathcal{T}$.*
2. *Let $A, B, C \in \mathcal{T} \cup \mathcal{E}$. Then, the graph D resulting from the merging of graphs A, B, C is also in $\mathcal{T}$.*

We will refer to a merging operation in $\mathcal{T}$ as a *generation step* or *merging step*. A sequence of generation steps leading to a graph in $\mathcal{T}$ shall be called *generation sequence*. Each generation step in a generation sequence has a different merging triple, which unequivocally identifies the step.

A generation sequence can be abstracted as a rewriting system, [64, 65], where objects are graphs and the generation steps are the rewriting rules. In this context, we will denote the application of a generation step merging clusters G, G_1, G_2 to the graph G as $G \rightarrow_{(G,G_1,G_2)} G^*$. If the hinges of the generation step are (u, v, w) we will also denote the application of the generation step as $G \rightarrow_{(u,v,w)} G^*$. The application of a generation sequence to K_3 leading to G shall be denoted by $K_3 \rightarrow_* G$. In order to distinguish between different generation sequences, we will use a supra index.

We prove now that $\mathcal{T}$ characterizes the set of tree-decomposable Laman graphs.

Theorem 7.1.1
Let $\mathcal{T}$ be the set of graphs defined in Definition 7.1.3, and TL the set of tree-decomposable Laman graphs. Then, $\mathcal{T} = TL$.

Proof
We prove first that any graph in $\mathcal{T}$ is Laman and tree-decomposable. We prove it by induction on the order of the graph.

Induction basis: Consider the first element in $\mathcal{T}$, K_3, with order 3. K_3 is trivially Laman and tree-decomposable.

Induction hypothesis: every graph in $\mathcal{T}$ with order less than n is Laman and tree-decomposable.

Consider $G \in \mathcal{T}$, with order n. We prove that G is Laman and tree-decomposable. Assume that G is the graph resulting from the merging of graphs $A = (V_A, E_A), B = (V_B, E_B)$ and $C = (V_C, E_C)$, all of them in $\mathcal{T}$. The order of graphs A, B, C is less than n, and, by induction hypothesis, A, B, C are Laman. Thus, the following relations hold:

$$|E_A| = 2|V_A| - 3, |E_B| = 2|V_B| - 3, |E_C| = 2|V_C| - 3$$

That is

$$|E_A| + |E_B| + |E_C| = 2(|V_A| + |V_B| + |V_C|) - 9$$

But $|E_G| = |E_A|+|E_B|+|E_C|$ and $|V_G| = |V_A|+|V_B|+|V_C|-3$, which yields $|E_G| = 2|V_G|-3$, and therefore G is Laman.

The same rational applies to prove the subgraph condition.

We prove now that G is tree-decomposable. By induction hypothesis, A, B, C are tree-decomposable. Since $G \in \mathcal{T}$ is the merging of graphs A, B, C, there exist $a, b, c \in V_G$ such that $V_A \cap V_B = \{a\}$, $V_A \cap V_C = \{b\}$, $V_B \cap V_C = \{c\}$. Then A, B, C define a tree-decomposition of graph G. The fact that A, B, C are tree-decomposable completes the proof.

We prove now that any tree-decomposable Laman graph G is a graph in $\mathcal{T}$. Let T_G be a tree-decomposition of G. The bottom-up merging of nodes in T_G is a generation sequence for G. Therefore $G \in \mathcal{T}$. □

We call the *corresponding generation step* of a tree-decomposition step T_G to the merging step having as merging triple the hinges of T_G. Analogously, we call the *corresponding tree-decomposition step* of a generation step S_G to the tree-decomposition step having as hinges the merging triple of S_G. Notice that, in this way, a generation sequence leading to a tree-decomposable Laman graph G is the inverse process of a tree-decomposition beginning at G.

Generating a graph G by means of a generation sequence assures the tree-decomposition of G by the tree-decomposition with the corresponding steps of the merging steps. In fact, we can establish a one-to-one correspondence between the set of generation sequences

leading to G and the tree-decompositions of G. We call the *corresponding generation sequence* of a tree-decomposition T_G to the generation sequence related by this bijection to T_G. Analogously, we call the *corresponding tree-decomposition* of a generation sequence S_G to the tree-decomposition related by this bijection to S_G. The existence of the above mentioned bijection proves the fact that the generation of graphs by means of a generating sequences is equivalent to tree-decomposability, but in bottom-up direction.

Theorem 7.1.2
The generation sequence leading to a tree-decomposable Laman graph is canonical.

Proof
In [64], Joan-Arinyo and Soto proved that tree-decompositions are canonical, that is, the same tree-decomposition steps appear in each one of the different tree-decompositions of a graph. Consider a graph G and a generation sequence S_G leading to it. Consider the corresponding tree-decomposition of S_G, T_G. T_G is canonical, and thus equivalent to other tree-decompositions of G, as T'_G. The corresponding generation sequence of T'_G, S'_G, is also a generation sequence leading to G, and the merging steps are the same as in S_G. Then, the generation sequence is also canonical. □

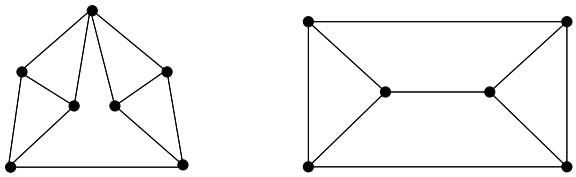
A consequence of Theorem 7.1.2 is that the set of merging triples of different generation sequences of a given graph is unique. Another consequence of this theorem is that the set of hinge triples and the set of merging triples of a graph is exactly the same. Moreover, hinges and merging vertices are two different names for the same concept. Then, we will identify from now on hinges and merging vertices.

7.1.3 Inclusion relations

The aim of this section is to establish the relation between H_I and H_{II} families of graphs and the set of tree-decomposable Laman graphs, characterized by $\mathcal{T}$. In particular, we will prove that $H_I \subset \mathcal{T} \subset H_{II}$. Each inclusion will be proved separately. The first one, $H_I \subset \mathcal{T}$ is a straightforward verification derived from the definition of the set $\mathcal{T}$.

Theorem 7.1.3
Let H_I be the set of graphs defined by a Henneberg sequence including only Henneberg I steps. Let $\mathcal{T}$ be the set defined in Definition 7.1.3. Then, $H_I \subset \mathcal{T}$.

Proof
The application of a HS1 to a graph G is equivalent to the merging of graph G with two edge graphs in $\mathcal{E}$, see Figure 7.1. The fact that H_I and $\mathcal{T}$ are different is proven by giving a counterexample. Graph in Figure 7.4a is tree-decomposable and thus belongs to $\mathcal{T}$. But it cannot be constructed with a sequence of HS1 steps because it does not include vertices of degree smaller than three. Otherwise, according to the definition of HS1, the graph would



a b

Figure 7.4: Counterexamples. a) Tree-decomposable Laman graph which cannot be constructed using only HS1. b) Non tree-decomposable Laman H_{II} graph.

have degree 2 vertices. □

For the inclusion $\mathcal{T} \subset H_{II}$ we need first a previous result on Henneberg graphs. The proof can be seen for example in [34].

Lemma 7.1.4
Let $G = (V, E)$ be a Laman graph. For any pair of vertices $\{u, v\} \subset V$, there is always a Henneberg sequence such that builds G from the graph $G_{(u,v)} \in \mathcal{E}$.

We can now prove the second inclusion.

Theorem 7.1.5
Let H_{II} be the set of graphs defined by a Henneberg sequence including Henneberg steps of type I and II. Let $\mathcal{T}$ be the set defined in Definition 7.1.3. Then, $\mathcal{T} \subset H_{II}$.

Proof
We prove first that every graph in $\mathcal{T}$ belongs to H_{II}. The inclusion is proven by induction on the order of the graph.

Induction base: Consider the first element in $\mathcal{T}$, K_3, with order 3. K_3 is in H_{II} by definition.

Induction hypothesis: Assume that every graph in $\mathcal{T}$ with order d, with $3 \leq d \leq n$, belongs to H_{II}.

We prove now that the merging of three arbitrary graphs in $\mathcal{T} \cup \mathcal{E}$ also belongs to H_{II}. Let $A = (V_A, E_A), B = (V_B, E_B), C = (V_C, E_C)$ be three graphs with $a_1, b_1 \in V_A$, $c_1, a_2 \in V_B$ and $b_2, c_2 \in V_C$. Consider the merging of A, B, C by identifying $a_1 = a_2 = a$, $b_1 = b_2 = b$ and $c_1 = c_2 = c$. Each graph A, B, C in $\mathcal{T} \cup \mathcal{E}$ either is an edge graph or, by Lemma 7.1.4, has a Henneberg sequence starting from an edge graph of any pair of its elements.

In particular, for each graph A, B, C we can find a Henneberg sequence starting from

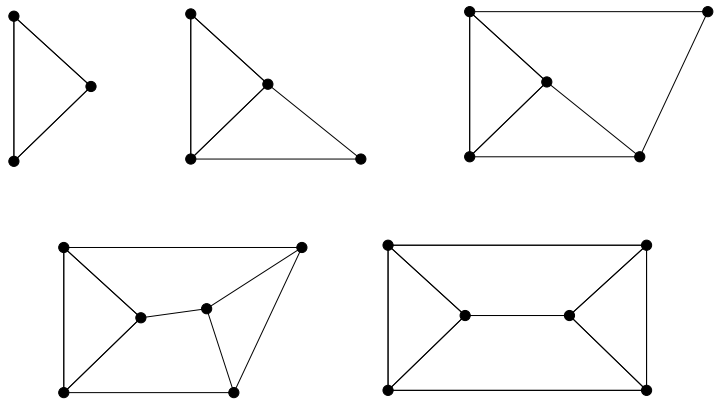


Figure 7.5: Henneberg sequence leading to the Laman graph in Figure 7.4b which is not tree-decomposable, represented from left to right and from top to bottom.

the two hinges included in it, and denote them respectively $G_{(a_1,b_1)} \Rightarrow_*^A A, G_{(a_2,c_1)} \Rightarrow_*^B B, G_{(b_2,c_2)} \Rightarrow_*^C C$. If A, B or C are in $\mathcal{E}$, the corresponding sequence will be just the edge graph. Now change the name of elements a_1, a_2 to a, b_1, b_2 to b and c_1, c_2 to c in A, B, C and the Henneberg sequences leading to them.

Consider now the triangle graph K_3 with vertices a, b, c. The Henneberg sequence resulting after applying Henneberg sequences $\Rightarrow_*^A, \Rightarrow_*^B, \Rightarrow_*^C$ to each one of the edges of K_3 leads to the merging of graphs A, B, C by identifying $a_1 = a_2 = a$, $b_1 = b_2 = b$ and $c_1 = c_2 = c$.

Next we show that $\mathcal{T} \neq H_{II}$. The fact that both sets are different is again proven by giving a counterexample: graph in Figure 7.4b is Laman, as can be seen in Figure 7.5, but it is not tree-decomposable, because there is no triple of vertices decomposing the graph in three subgraphs sharing pairwise just one vertex. □

7.2 Preserving tree-decomposability in Henneberg steps

It is well known, and it has been also proved in Section 7.1, that not all Laman graphs are tree-decomposable. For that reason, when building a Laman graph as a Henneberg sequence, the resulting graph may or may not be tree-decomposable. The traditional way to figure out whether a Laman graph is tree-decomposable is to proceed to a decomposition and then check if it has succeeded. We devote this section to establish the conditions under which a Henneberg sequence results in a tree-decomposable Laman graph. We will analyze tree-decomposability preservation for each kind of Henneberg step.

7.2.1 Henneberg I steps and tree-decomposition

Proving tree-decomposability preservation of Henneberg I steps is straightforward, and can be seen as a Corollary of Theorem 7.1.3.

Corollary 7.2.6
Let G and G^* be two Laman graphs such that $G \Rightarrow_1 G^*$. Then G^* is tree-decomposable if and only if G is tree-decomposable.

Proof
For the if part, consider that G is tree-decomposable. Then, there exists a generation sequence S from K_3 to G. As seen in Theorem 7.1.3, the application of a HS1 to a graph G is equivalent to the merging of graph G with two edges in $\mathcal{E}$. Then, we can define a generation sequence from K_3 to G^* by adding to S a last generation step related to the application of the HS1.

For the only if part, assume that $G = (V, E)$ and $G^* = (V^*, E^*)$ is tree-decomposable. Since $G \Rightarrow_1 G^*$, there exist two vertices $v_1, v_2 \in V$ and a vertex $v \in V^*$ such that $V^* = V \cup \{v\}$ and $E^* = E \cup \{(v_1, v), (v_2, v)\}$. Consider the tree-decomposition step with hinges v, v_1, v_2 which decomposes G^* in threee subgraphs: G, $G_{(v_1,v)}$ and $G_{(v_2,v)}$. Since G^* is tree-decomposable, the subgraphs are tree-decomposable and in particular G is tree-decomposable. □

7.2.2 Henneberg II steps and tree-decomposition

Proving tree-decomposability preservation of Henneberg II steps is not trivial. Before addressing it, we present some basic features of the tree-decomposition process of tree-decomposable graphs. The first result states that every vertex in a tree-decomposable graph is a hinge at least in one tree-decomposition step.

Lemma 7.2.7
Let $G = (V, E)$ be a tree-decomposable Laman graph. Then every vertex $v \in V$ is hinge in at least one tree-decomposition step. Moreover, for each edge $e = (v_1, v_2)$ in E, there exists at least one tree-decomposition step including vertices v_1, v_2 as hinges.

Proof
In [106], Vila showed that to each edge $e = (v_1, v_2)$ in G corresponds a leaf $\{v_1, v_2\}$ on the tree-decomposition of G, which means that every edge is a cluster in at least one tree-decomposition step. In the tree-decomposition step where e is one of the clusters, vertices v_1, v_2 are hinges. Besides, as there are no degree 0 vertices in a tree-decomposable Laman graph, every vertex v has at least one incident edge. □

Now we state and prove a Lemma concerning the nature of a cluster in which a hinge has degree 1.

Lemma 7.2.8
Let $G = (V, E)$ be a tree-decomposable Laman graph made of the merging of clusters G_1, G_2, G_3 by means of hinges u, v, w, with $G_1 = (V_1, E_1)$ and $v \in V_1$. If v has degree 1 in G_1, then $G_1 \in \mathcal{E}$.

Proof
Consider cluster G_1, vertex $v \in V_1$ and the degree of v in G_1 is 1. Then, G_1 can not be Laman. By definition, cluster $G_1 \in \mathcal{T} \cup \mathcal{E}$. Since $G_1 \notin \mathcal{T}$, then it must be $G_1 \in \mathcal{E}$. □

Finally we need to define the concept of maximal rigid subgraph of an under constrained graph, since it will be used later on.

Definition 7.2.1
Let G be a tree-decomposable under-constrained graph. Consider a tree-decomposable Laman subgraph R of G. R is a maximal tree-decomposable Laman subgraph of G if there exists no other tree-decomposable Laman subgraph M such that $R \subset M \subset G$. The maximal tree-decomposable Laman subgraph of a Laman graph G is G itself.

Notice that a maximal rigid subgraph in an under constrained graph does not need to be unique.

Now we will prove two different theorems stating the conditions for the tree-decomposability preservation of Henneberg II steps. The first result explains why some constructions give rise to non tree-decomposable graphs, while the second one gives a framework for the development of an efficient and correct algorithm to build tree-decomposable Laman graphs. The first theorem is illustrated in Figure 7.6.

Theorem 7.2.9
Let $G = (V, E)$ be a tree-decomposable Laman graph and G^* the graph such that $G \Rightarrow_2 G^*$, where the applied Henneberg II step involves edge $e = (v_1, v_2) \in E$ and vertex $v_3 \in V$. Then, G^* is tree-decomposable if, and only if, there exists a generation step of a generation sequence of G merging clusters G_1, G_2, G_3 such that $G_1 \in \mathcal{E}$, $e \in E(G_1)$, v_1, v_2 are hinges of S and v_3 is in G_2, in G_3 or in both of them.

Proof
Let v denote the new vertex involved in the Henneberg II step. We first prove the case in which v_3 is in just one of the clusters G_2 or G_3. Without loss of generality, we assume that $v_3 \in G_2$, that is, the situation is the one depicted in Figure 7.6a. At the end of the proof we will consider the case where v_3 is in both clusters, depicted in Figure 7.6c.

For the if part, assume that there exists a generation sequence fulfilling the requirements

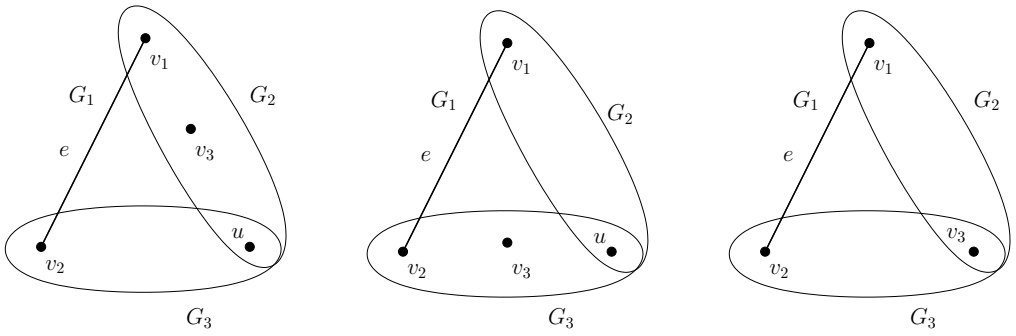


Figure 7.6: Necessary tree-decomposition step for the preservation of the tree-decomposability in Henneberg II steps, Theorem 7.2.9.

of the statement of the theorem, that is, $K_3 \rightarrow_*^i G_i \rightarrow_*^1 G$, where G_i is the merging of clusters G_1, G_2, G_3 such that $G_1 \in \mathcal{E}$, $e \in E(G_1)$ and v_1, v_2 are hinges. Then, if G_{i-1} is such that $K_3 \rightarrow_*^{i-1} G_{i-1} \rightarrow_{(G_1,G_2,G_3)} G_i \rightarrow_*^1 G$, G_{i-1} must be one of the clusters G_1, G_2 or G_3. If $G_{i-1} \neq G_2$, consider a generation sequence of G_2, $K_3 \rightarrow_*^0 G_2$. Then, $K_3 \rightarrow_*^0 G_2 \rightarrow_{(G_1,G_2,G_3)} G_i \rightarrow_*^1 G$ is a generation sequence of G.

Now, by hypothesis, $v_3 \in G_2$, and either v_1 or v_2 is also in G_2. Assume $v_1 \in G_2$, and define $A = G_{(v_1,v)} \in \mathcal{E}$ and $B = G_{(v_3,v)} \in \mathcal{E}$. The merging of graphs G_2, A and B, denoted $G_2 \rightarrow_{(G_2,A,B)} G_2^*$, gives rise to graph G_2^* depicted in Figure 7.7a. Now define $C = G_{(v_2,v)} \in \mathcal{E}$. The merging of graphs G_2^*, G_3 and C, denoted $G_2^* \rightarrow_{(G_2^*,G_3,C)} G_2^{**}$, gives rise to graph G_2^{**} depicted in Figure 7.7b. Then,

$$K_3 \rightarrow_*^0 G_2 \rightarrow_{(G_2,A,B)} G_2^* \rightarrow_{(G_2^*,G_3,C)} G_2^{**} \rightarrow_*^1 G^*$$

is a generation sequence of G

For the only if part, assume that G^* is a tree-decomposable Laman graph, $K_3 \rightarrow_* G^*$. Since $v \in V(G)$, there exists a graph G_i in the generation sequence in which v appears for the first time. If $\deg(v)>2$ in G_i, consider the tree-decomposition of the cluster in which v has greater degree. Applying the same rule, and considering that $\deg(v)$ in G^* is 3, we finally identify a generation sequence

$$K_3 \rightarrow_*^0 G_j^* \rightarrow_*^1 G_k^* \rightarrow_*^2 G^*$$

where G_j^* is the first graph including v, $\deg(v)$ in G_j^* is 2, G_k^* is the first graph in which $\deg(v)=3$, and the hinges in $\rightarrow_*^2$ do not include v.

The graph G_k^* is the merging of three clusters, say G_1^k, G_2^k, G_3^k, and is the first in which $\deg(v)=3$. Then, v is hinge of that merging step, and therefore the only vertex in the

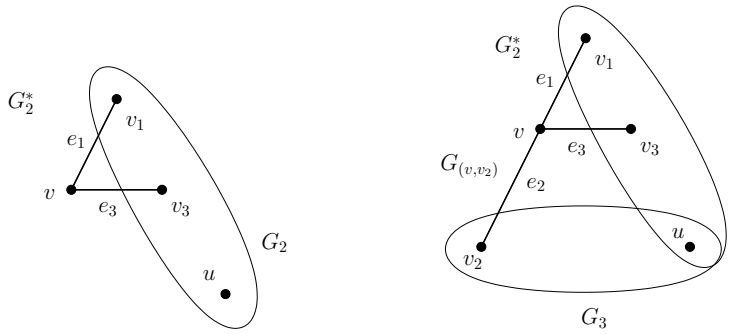


Figure 7.7: Illustration of Theorem 7.2.9. a) Graph G_2^* resulting after the merging of graph G_2 with two edge graphs. b) Graph G_2^{**} resulting after the merging of graph G_2^* with an edge graph and G_3.

intersection of two clusters. Consider $G_1^k \cap G_2^k = \{v\}$. Since $\deg(v)$=3 in G_k^*, in one of the two clusters $\deg(v)$=1, say G_1^k. By Lemma 7.2.8, $G_1^k \in \mathcal{E}$. Since v is joined to vertices v_1, v_2, v_3, one of them must be in G_1^k, being also hinge of S, and the other two are in G_2^k, as v. The schematic representation of this step is shown in Figure 7.8. We consider two cases, the one in which $v_3 \in G_1^k$, and the one in which $v_3 \in G_2^k$.

First consider the case in which $v_3 \in G_1^k$. Then, $v_1, v_2 \in G_2^k$, as shown in Figure 7.8a. Take G and perform the tree-decomposition steps corresponding to the generation sequence $\rightarrow_*^2$, which can be done because they do not contain v as hinge and all the other vertices are included in G. The resulting graph is depicted in Figure 7.9a, and can not be Laman. This is a contradiction and the case can not actually occur.

Now consider the case in which $v_3 \in G_2^k$. Then, v_1, v_2 are in different clusters. Take G and perform the tree-decomposition steps corresponding to the generation sequence $\rightarrow_*^2$. The resulting graph, G_k, is depicted in Figure 7.9b and fulfills the requirements in the statement of the theorem. Cluster G_2 is such that the application of a HS1 results in G_2^k, and by Corollary 7.2.6 it is tree-decomposable. Then, there exists a generation sequence $K_3 \rightarrow_*^1 G_2$. The generation sequence $K_3 \rightarrow_*^1 G_2 \rightarrow_{(v_1,v_2,u_3)} G_k \rightarrow_*^2 G$ fulfills then the theorem.

For the case where v_3 is in clusters G_2 and G_3, as illustrated in Figure 7.6c, the proof can be done exactly in the same way. As proven above, in the only if part of the proof v_3 can not be in cluster G_1^k. This observation completes the proof. □

Theorem 7.2.9 establishes the conditions under which a Henneberg II step preserves tree-decomposability. However, to verify whether these conditions hold cannot be made

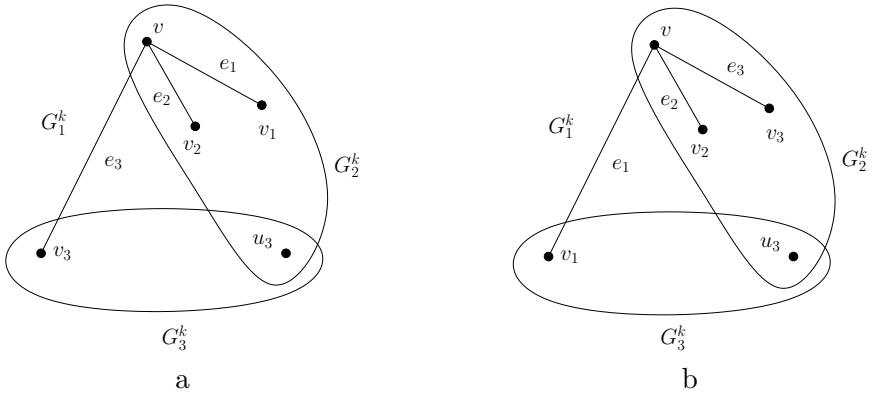


Figure 7.8: Illustration of Theorem 7.2.9. a) Case in which v_1, v_2 are in the same cluster. b) Case in which v_1, v_2 are in different clusters.

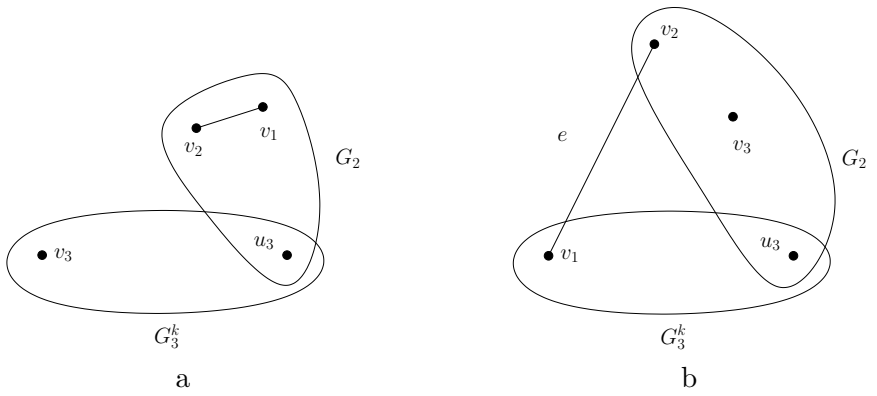


Figure 7.9: Illustration of Theorem 7.2.9. a) A graph G which does not fulfill Laman condition. b) Tree-decomposable Laman graph G.

efficiently because in the worst case it entails to check all possible generation steps.

We present now a new theorem which provides the tools for an efficient algorithm for checking the tree-decomposability. The proof is based on the previous result.

Theorem 7.2.10
Let $G = (V, E)$ be a tree-decomposable Laman graph. Let G^* be the graph such that $G \Rightarrow_2 G^*$, and edge $e = (v_1, v_2) \in E$ and vertex $v_3 \in V$ be the edge and vertices involved in the HS2. Then, G^* is tree-decomposable if and only if, either

- *(v_1, v_2, v_3) is a merging triple of G or,*
- *when removing edge e from E, there exists one maximal tree-decomposable Laman subgraph $R = (V_R, E_R)$ of G such that either $\{v_3, v_1, u\} \subseteq V_R$ or $\{v_3, v_2, u\} \subseteq V_R$, and u is such that (u, v_1, v_2) is a hinge triple of G.*

Proof
For the only if part, consider that G^* is tree-decomposable. By Theorem 7.2.9, there exists a generation sequence of G such that, for some generation step S merging clusters G_1, G_2, G_3, edge e is the only edge in cluster G_1, v_1 and v_2 are merging vertices of S and v_3 belongs either to G_2, to G_3 or to both.

In the case that v_3 belongs to both clusters G_2 and G_3, the situation is the one depicted in Figure 7.6c and there exists obviously a triple of hinges v_1, v_2, v_3.

We consider now the cases in which v_3 belongs only to one cluster. We call u the third hinge of the tree-decomposition step including v_1, v_2 as merging vertices. Assume without loss of generality that $v_3 \in V(G_2)$, see Figure 7.6a. When removing edge e from E to apply the HS2 and transform G in G^*, cluster G_2 does not change, since $e \notin E(G_2)$. Then, G_2 will be contained in a maximal Laman subgraph R. Thus, $v_1, v_3, u \in V(G_2) \subseteq V_R$, proving the first part of the theorem.

For the if part, consider first the case in which (v_1, v_2, v_3) is a merging triple of G. Then, there exists a generation sequence $K_3 \rightarrow_*^0 G_k \rightarrow_{(v_1,v_2,v_3)} G_{k+1} \rightarrow_*^1 G$, where G_1 is the name of the cluster in generation step $\rightarrow_{(v_1,v_2,v_3)}$ such that $v_1, v_2 \in G_1$. Since edge $(v_1, v_2) \in E$, every step in G_1 is either independent or directly dependent on the generation step $\rightarrow_{(v_1,v_2,v_3)}$. Then, every step in G_1 can be performed before the generation step $\rightarrow_{(v_1,v_2,v_3)}$. There exists then a generation sequence $K_3 \rightarrow_*^2 G_j \rightarrow_{(v_1,v_2,v_3)} G_{j+1} \rightarrow_*^3 G$, where the cluster including vertices v_1, v_2 is $G_{(v_1,v_2)}$.

Call G_2 to the cluster in the generation step $\rightarrow_{(v_1,v_2,v_3)}$ including vertices v_1, v_3, and G_3 to the cluster including vertices v_2, v_3, which are Laman and tree-decomposable. Call $A = G_{(v,v_1)} \in \mathcal{E}$ and $B = G_{(v,v_3)} \in \mathcal{E}$. Consider the graph G_2^* resulting after the merging of clusters G_2, A and B, shown in Figure 7.10a. Call now $C = G_{(v,v_2)} \in \mathcal{E}$, and G_2^{**} the

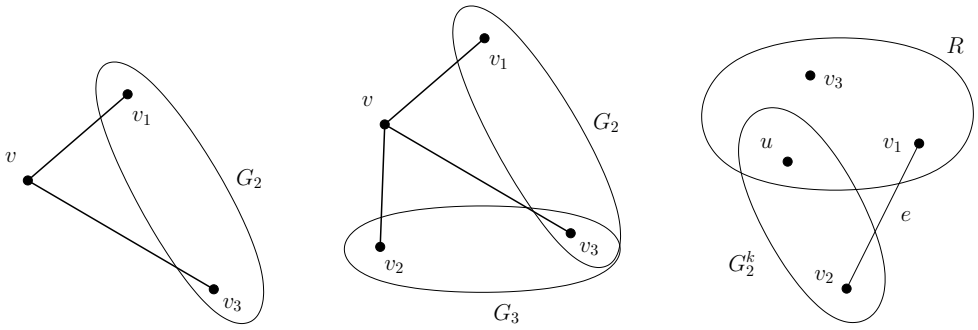


Figure 7.10: Illustration of Theorem 7.2.10. a) Graph resulting after the merging of G_2 with two edge graphs. b) Tree-decomposition step of a graph after the application of a HS2 involving elements v_1, v_2, v_3. c) Construction of G by applying a HS1 adding vertex v_2.

graph resulting after the merging of clusters G_2^*, G_3 and C, shown in Figure 7.10b. Then, $K_3 \rightarrow_* G_2 \rightarrow_{(G_2,A,B)} G_2^* \rightarrow_{(G_2^*,G_3,C)} G_2^{**} \rightarrow_*^1 G^*$ is a generation sequence of G^*.

Consider now the case in which (v_1, v_2, v_3) is not a merging triple of G and that, when removing edge e from E, there exists a maximal tree-decomposable Laman subgraph R like the one described in the statement of this theorem. Consider without loss of generality that $v_1 \in R$. Since (u, v_1, v_2) is a merging triple of G, there exists a generation sequence of G with $K_3 \rightarrow_*^0 G_k \rightarrow_{(v_1,v_2,u)} G_{k+1} \rightarrow_*^1 G$, where G_{k+1} is the merging of three clusters. Call G_2^k the cluster including vertices u, v_2.

Since R is a tree-decomposable Laman graph, there exists a generation sequence $K_3 \rightarrow_*^r R$. Define the graph $A = G_{(v_1,v_2)} \in \mathcal{E}$, and consider the merging of graphs R, G_2^k and A. The resulting graph G_j is shown in Figure 7.10c. Since G_j is a tree-decomposable Laman subgraph of G, there exists a generation sequence such that $G_j \rightarrow_*^j G$. Then $K_3 \rightarrow_*^r R \rightarrow_{(G_2^k,A,R)} G_j \rightarrow_*^j G$ is a generation sequence of G fulfilling the requirements in Theorem 7.2.9, and thus G^* is tree-decomposable. □

Theorem 7.2.10 provides a condition which leads to a more efficient way to generate tree-decomposable Laman graphs, as it removes the necessity of checking all possible tree-decompositions in the worst case.

With this result, we can easily prove that for every edge e removed in a HS2, it exists at least one vertex for which the application of this HS2 results in a tree-decomposable graph. That means that there are tree-decomposable graphs of any order. We will explain this point in depth in Section 7.3.

Corollary 7.2.11
Let $G = (V, E)$ be a tree-decomposable Laman graph, and $e = (v_1, v_2) \in E$. Then, there exists always a vertex $v_3 \in V$ such that the application of a Henneberg II step to G involving e and v_3 leads to a tree-decomposable Laman graph.

Proof
Given edge e defined upon vertices v_1 and v_2, by Lemma 7.2.7, there exists at least one generation step with hinges v_1 and v_2. This generation step must include also a third hinge, u. To consider $v_3 = u$ proves the corollary. □

Clearly, a Henneberg sequence beginning at K_3 such that all its Henneberg steps preserve tree-decomposability leads to a tree-decomposable Laman graph.

7.3 An algorithm to generate tree-decomposable graphs

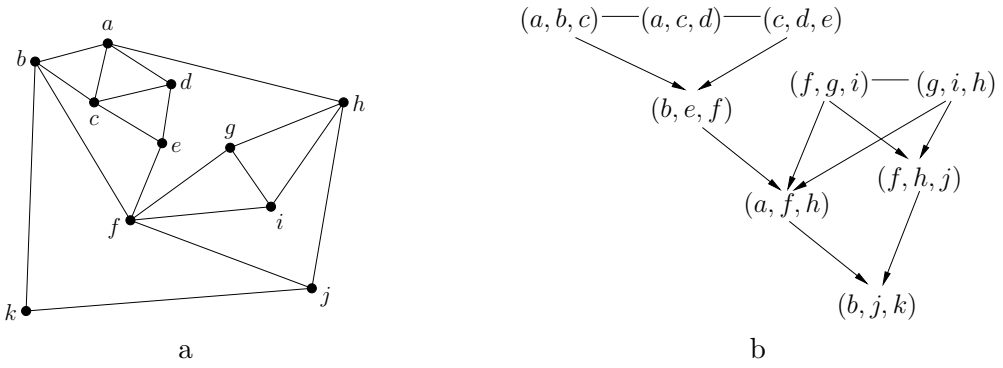
We present in this section an algorithm based on h-graphs to generate tree-decomposable graphs of a given order by means of Henneberg sequences. We assure tree-decomposability by applying the results proven in Section 7.2. We also show that the presented algorithm is correct.

The general idea of the algorithm is to consider as a base the triangle graph $G = K_3$ and the associated h-graph $\mathcal{H}(G)$, defined in Chapter 3, and then to apply an arbitrary Henneberg sequence, assuring tree-decomposability and conveniently updating the associated h-graph. The h-graph will help in the efficient computation of a vertex assuring the tree-decomposability of the resulting graph.

Before presenting the proposed algorithm we address some issues about h-graphs. In particular, we shall show how to construct h-graphs from the Henneberg sequence of a graph, and how to compute the set of maximal tree-decomposable Laman subgraphs included in an under constrained tree-decomposable graph.

7.3.1 Henneberg constructions and h-graphs

Given a tree-decomposable Laman graph G and the associated h-graph $\mathcal{H}(G)$, the application of different Henneberg steps to G will lead to changes in the structure of G. In this section we analyze how the h-graph $\mathcal{H}(G)$ changes when applying a Henneberg step construction to G. We distinguish two cases depending on whether the applied Henneberg step is HS1 or HS2.



a b

Figure 7.11: Application of a HS1. a) Resulting graph G^* after adding vertex k and edges $(b,k),(k,j)$ to the graph in Figure 3.3a. b) h-graph $\mathcal{H}(G^*)$ associated to G^*.

Adding a Henneberg I step

Consider the tree-decomposable Laman graph G and its associated h-graph $\mathcal{H}(G)$. Consider the tree-decomposable graph G^* such that $G \rightarrow_{H1} G^*$, where the added vertex is v and is joined by means of edges (v_1, v) and (v_2, v) to vertices v_1, v_2, see Figure 7.1.

In order to define the h-graph $\mathcal{H}(G^*)$ associated to G^* it suffices to add to $\mathcal{H}(G)$ the node $V = (v_1, v_2, v)$ and its corresponding dependences. We will explain now how to compute the vertices to which vertex V will be joined.

By the application of the HS1, $G^* = (V^*, E^*)$ contains edges (v_1, v) and (v_2, v), see Figure 7.1. If E^* contains also the edge (v_1, v_2), by Lemma 7.2.7, there exists in $\mathcal{H}(G)$ at least one node including v_1 and v_2 as hinges. Then, the node V must be joined with a d-edge with every node in $\mathcal{H}(G)$ including v_1 and v_2 as hinges.

If E^* does not include the edge (v_1, v_2), V is joined with s-edges to the representative vertices of the minimum subgraph spanned by the elements v_1 and v_2. The minimum subgraph and its representative vertices are computed as explained in Section 3.5.

Algorithm 12 shows the method to update the h-graph $\mathcal{H}(G)$ associated to a tree-decomposable Laman graph G when a HS1 is applied to G. To illustrate the idea, Figure 7.11 shows the result of the application of a HS1 step adding vertex k and edges (b,k) and (j,k) to the graph G depicted in Figure 3.3a.

Algorithm 12 Add HS1

Input: G = (V, E), the tree-decomposable Laman graph,
HG = (HV, ED, ES), the h-graph associated to the graph G,
v, the new vertex in G
v1, v2, the vertices to which v is joined
Output: The modified graph HG

function Add_HS1()
V_0 := (v1, v2, v)
HV = HV ∪ { V_0 }
if E contains (v1, v2) **then**
L := Set of nodes in HV including v1, v2
for each vertex L_i in L **do**
ED = ED ∪ { (L_i, V_0) }
end for
else
R:= Compute_Strong_Dependences(G, v1, v2)
for each node R_i in R **do**
ES = ES ∪ { (R_i, V_0) }
end for
end if
return (HV, ED, ES)

Adding a Henneberg II step

Consider a tree-decomposable Laman graph $G = (V, E)$ and the associated h-graph $\mathcal{H}(G)$. Let G^* be the graph such that $G \rightarrow_{II} G^*$. As seen in Section 7.2, there is no guarantee that the resulting graph G^* is a tree-decomposable Laman graph. However, if G^* is a tree-decomposable Laman graph, we can define an algorithm based on Theorem 7.2.10 which returns the h-graph associated to it. If G^* is not tree-decomposable, the algorithm fails. This algorithm is shown in Algorithms 13 and 14.

Algorithm 13 Add nodes

```
Input: G = ( V, E ), the tree-decomposable Laman subgraph,
       HG = ( HV, ED, ES ), the h-graph associated to G
       v, the new vertex in G
       v1, v2, v3, the vertices involved in the HS2
Output: The modified graph HG

function Add_Nodes()
L := Vertices in { v1, v2, v3 } included also in a node of HV
if L contains v1 and L contains v3 then
  HG = add_HS1 ( G, HG, v, v1, v3 )
  HG = add_HS1 ( G, HG, v, v2, v3 )
else if L contains v2 and L contains v3 then
  HG = add_HS1 ( G, HG, v, v2, v3 )
  HG = add_HS1 ( G, HG, v, v1, v3 )
else if L contains v1 and L contains v2 then
  V1 := ( v1, v3, v )
  V2 := ( v2, v3, v )
  HV = HV ∪ { V1, V2 }
  ED = ED ∪ { ( V1, V2 ) }
  L := Set of Nodes in HV including v1, v2
  for each vertex Li in L do
    ES = ES ∪ { ( V1, Li ) }
    ES = ES ∪ { ( V2, Li ) }
  end for
end if
return HG
endfunction
```

For an example, we consider the application of a HS2 step to the graph G depicted in Figure 3.3a. Edge (a, h) is removed and the vertex k and the edges $(a, k), (k, h)$ and (k, g) are added. The elements involved, following the notation in this section are $v_1 = a, v_2 =$

Algorithm 14 Add HS2

```
Input: G = ( V, E ), the tree-decomposable Laman subgraph,
       HG = ( HV, ED, ES ), the h-graph associated to G
       v, the new vertex in G
       v1, v2, v3, the vertices involved in the HS2
Output: The modified graph HG

function Add_HS2()
V0 := Node in HV including hinges v1, v2, v3
Remove Not Existing Nodes from HG ( v1, v2, v3 )
L:= Set of Maximal Complete Subgraphs included in HG
if V0 != NULL then
  HS := Subgraph in L including two of the vertices v1, v2, v3
  S := Tree-decomposable Laman graph associated to HS
  HS = add_Nodes ( S, HS, v, v1, v2, v3 )
  HG = Add to HS the rest of subgraphs in L
else
  HS := Subgraph in L including v1 or v2, v3 and u, where ( v1, v2, u ) in HV
  S := Tree-decomposable Laman graph associated to HS
  if HS != NULL then
    Rename the vertex included in HS as v1
    Rename the vertex not included in HS as v2
    HS = add_HS1( S, HS, v, v1, v3 )
    HS = add_HS1( S, HS, v2, u, v )
    HG = Add to HS the rest of subgraphs in L
  end if
end if
return HG
endfunction
```

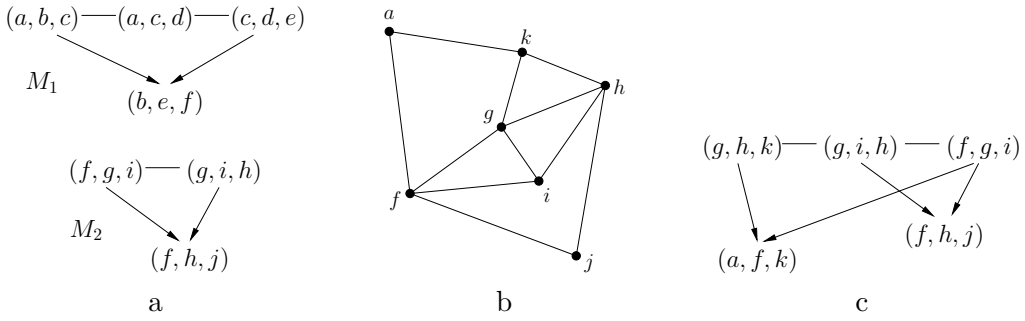


Figure 7.12: Application of a HS2. a) h-graphs associated to the remaining maximal tree-decomposable Laman subgraphs M_1, M_2. b) Graph M_2' after the application of the two indicated HS1. c) h-graph associated to M_2', $\mathcal{H}(M_2')$.

$h, v_3 = g$ and $v = k$. The resulting graph G^* is shown in Figure 7.13a.

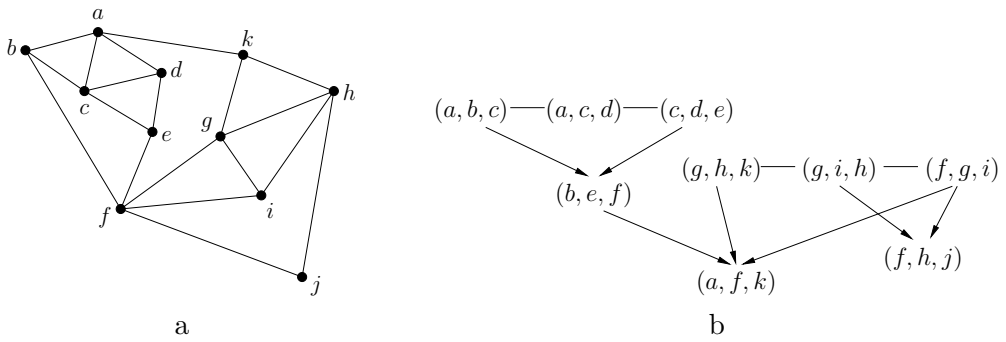
Algorithm 14 checks whether there exists the node (a, h, g), but the answer is negative. Node (a, h, f) including elements a and h is removed, and no other node is removed. The two remaining maximal rigid subgraphs M_1 and M_2 are the ones including vertices $\{a, b, c, d, e, f\}$ and $\{f, g, h, i, j\}$ respectively. Their respective associated h-graphs are shown in Figure 7.12a.

The maximal Laman subgraph fulfilling the conditions of Theorem 7.2.10 is M_2, as it contains vertices $v_3 = g, v_i = h$ and $u = f$. We rename the vertices according to the notation in this section, leading to $v_1 = h$, $v_2 = a$, $v_3 = g$ and $u = f$. Then the algorithm applies to M_2 two HS1 steps, adding k and a. The resulting graph is depicted in Figure 7.12b and the associated h-graph in Figure 7.12c. The merging of this graph with graph M_1 and $G_{(a,k)}$ gives rise to the desired graph G^*, depicted in Figure 7.13 together with the associated h-graph.

7.3.2 Maximal Laman subgraph in h-graphs

The maximal tree-decomposable Laman subgraph included in a tree-decomposable Laman graph once a set of its edges has been removed played a central role in Theorem 7.2.10, and will be basic in the method we will propose to generate tree-decomposable Laman graphs of a given order.

Consider a tree-decomposable Laman graph G and the graph G' resulting after removing from G some of its edges. We present in Algorithm 15 a method which, starting in a fixed node of the h-graph $\mathcal{H}(G')$ associated to the subgraph G', constructs the graph R by adding an adjacent node v to $\mathcal{H}(R)$ only if $\text{dep}_G(v) \subseteq \mathcal{H}(R)$, that is, only if the resulting graph



a b

Figure 7.13: Application of a HS2. a) Resulting graph G^* after removing edge (a, h) and adding vertex k and edges $(a, k), (g, k), (h, k)$ to the graph in Figure 3.3a. b) h-graph $\mathcal{H}(G^*)$ associated to G^*.

is complete. HG = (HV, ED, ES) stand for the h-graph of G, HG' = (HV', ED', ES') stand for the h-graph of G' and SHG = (SV, SED, SES) stand for the h-graph of R.

With this algorithm we construct a function which finds all the maximal tree-decomposable Laman subgraphs included in an under constrained tree-decomposable graph. When no vertex can be added to the current subgraph R in Algorithm 15, R is stored and another maximal tree-decomposable subgraph is computed.

7.3.3 The algorithm

Consider a tree-decomposable Laman graph $G = (V, E)$ and the associated h-graph $\mathcal{H}(G)$. We present in this section a method which allows to construct using Henneberg stepd tree-decomposable Laman graphs of any order. According to the results derived in Section 7.2, the application of a HS1 step preserves tree-decomposability, but this is not the case when the Henneberg step applied is a HS2.

Consider the application to G and $\mathcal{H}(G)$ of an arbitrary HS2 step involving edge $e = (v_1, v_2) \in E$ and $v_3 \in V$. In order to assure tree-decomposability of the resulting graph, a first approach could use the particular case in Corollary 7.2.11 to define always as the third vertex v_3 of the HS2 step involving edge (v_1, v_2) an element which is hinge in a triple together with vertices v_1 and v_2. Although the method guarantees tree-decomposability of the resulting graph and can be easily implemented, it can not be used to generate graphs in H_{II} since the resulting graph will be in H_I. The following result shows it.

Proposition 7.3.12
Let $G = (V, E)$ be a H_I graph and G^* be the graph such that $G \Rightarrow_2 G^*$, where the HS2 step

Algorithm 15 Find Maximal Complete Subgraph

```
Input: HG = ( HV, ED, ES ), the h-graph of G
       HG' = ( HV', ED', ES' ), the h-graph of G'
       V0, a node in HV included in the subgraph
Output: SHG = ( SV, SED, SES ), a maximal complete subgraph

function Find_Maximal_Complet_Subgraph()
SV := { V0 }
SED := ∅
SES := ∅
flag = TRUE
while flag do
  flag = FALSE
  for all V_i in SV do
    E := Edges in ED' including V_i
    for all E_j in E do
      V_j := Node in E_j opposite to V_i
      SV = SV ∪ { V_j }
      SED = SED ∪ { ( V_i, V_j ) }
      flag = TRUE
    end for
    E := Edges in ES' including V_i
    for all E_j in E do
      V_j := Node in E_j opposite to V_i
      D := Indirect dependences of V_j in HG
      if SV contains D then
        SV = SV ∪ { V_j }
        SES = SES ∪ { ( V_i, V_j ) }
        flag = TRUE
      end if
    end for
  end for
end while
return SHG = ( SV, SED, SES )
endfunction
```

involves edge $e = (v_1, v_2)$ and vertex $v_3 \in V$. If there exists a tree-decomposition step with hinges v_1, v_2, v_3, then $G^* \in H_I$.

Proof
Since $G \in H_I$, there exists a Henneberg sequence leading to G in which all the steps are HS1. Then, $K_3 \Rightarrow_{*1} G \Rightarrow_2 G^*$. By hypothesis, one of the HS1 steps has as hinges (v_1, v_2, v_3), which means, without loss of generality, that vertex v_1 is added joined to vertices v_2 and v_3. Therefore there exists a Henneberg sequence $K_3 \Rightarrow^1_{*1} G_i \Rightarrow^{(v_1,v_2,v_3)}_1 G_{i+1} \Rightarrow^2_{*1} G \Rightarrow_2 G^*$, where G_i is depicted in Figure 7.14 top.

Since all the Henneberg steps in $\Rightarrow^2_{*1}$ are HS1 and add a vertex and two edges to the previous graph, we can consider a Henneberg sequence of G^* in which the HS2 step is right after the HS1 step adding vertex v_1,

$$K_3 \Rightarrow^1_{*1} G_i \Rightarrow^{(v_1,v_2,v_3)}_1 G_{i+1} \Rightarrow_2 G_{i+2} \Rightarrow^3_{*1} G^*$$

Now we will find a Henneberg sequence for G^* such that all the steps are of type HS1. Consider the HS1 step $\Rightarrow^{(v,v_2,v_3)}_1$ adding vertex v and edges $(v, v_2), (v, v_3)$. Consider also the HS1 step $\Rightarrow^{(v_1,v,v_3)}_1$ adding vertex v and edges $(v_1, v), (v_1, v_3)$. Then, the Henneberg sequence

$$K_3 \Rightarrow^1_{*1} G_i \Rightarrow^{(v,v_2,v_3)}_1 G'_{i+1} \Rightarrow^{(v_1,v,v_3)}_1 G_{i+2} \Rightarrow^3_{*1} G^*$$

is a Henneberg sequence for G^* with HS1 steps. □

Figure 7.14 shows the construction of G^* by means of the original sequence (left), and by means of the alternative sequence with only HS1 steps (right). Notice that the two resulting graphs are the same. This result shows that the application of a HS2 to H_I graphs under the conditions of Proposition 7.3.12 gives rise to a H_I graph. Then, since $K_3 \in H_I$ is the first considered graph, generated graphs will be in H_I.

In order to assure tree-decomposability of the resulting graph we will define v_3 so that it guarantees that the HS2 step preserves tree-decomposability. We will do it by discarding all the vertices known to produce a non tree-decomposable resulting graph.

Following Theorem 7.2.10, the graph resulting after a HS2 step is tree-decomposable if there exists a maximal tree-decomposable Laman subgraph $R = (V_R, E_R)$ of G such that either $\{v_3, v_1, u\} \subseteq V_R$ or $\{v_3, v_2, u\} \subseteq V_R$, and u is such that (u, v_1, v_2) is a hinge triple of G. In fact, any vertex in R will act as a vertex v_3 assuring tree-decomposability.

In order to search for that subgraph R, the first step is to find in $\mathcal{H}(G)$ the set L of all the nodes including v_1 and v_2. Then, we remove from $\mathcal{H}(G)$ all the nodes having any of the nodes in L as dependence. Then, the graph is split into a set of complete subgraphs. The nodes in L have not been removed, and must be included in one of the complete maximal

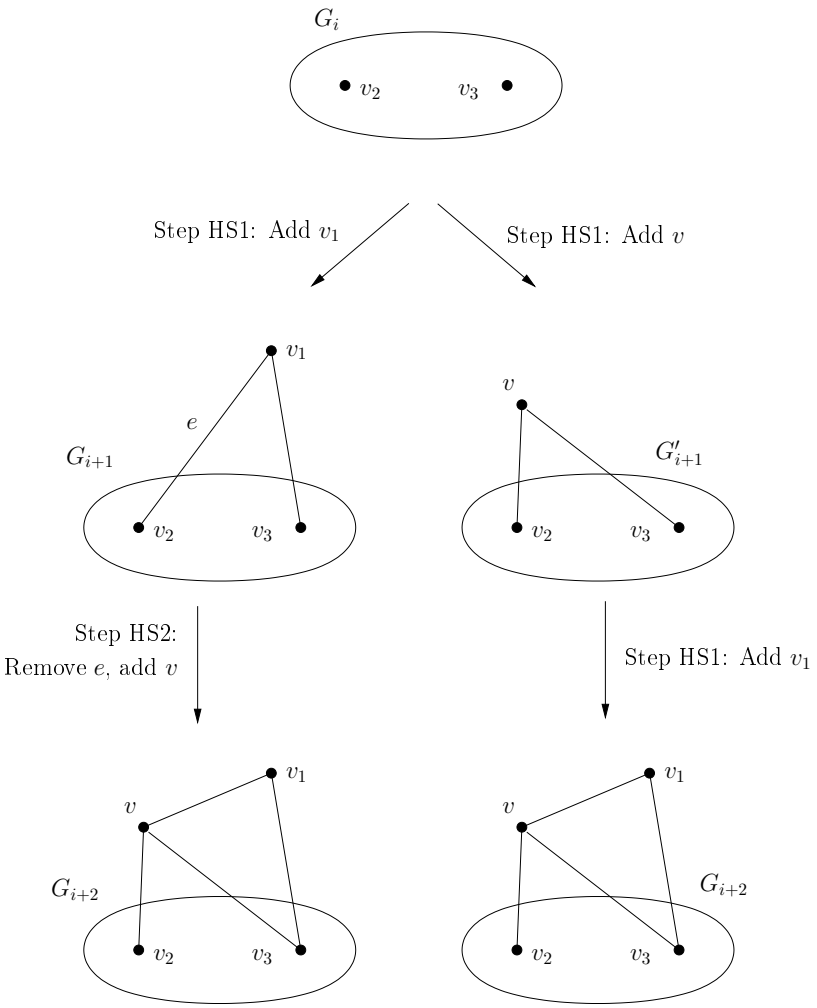


Figure 7.14: Construction of G' by means of the original Hennberg II step (left), and by means of the alternative Henneberg I sequence (right).

subgraphs, say M. Any vertex in a node in M different to v_1 and v_2 can be selected as vertex v_3 and will assure tree-decomposability.

Algorithms 16, 17, 18 and 19 are the implementation of the method which computes a tree-decomposable Laman graph with a predefined order. The main algorithm is shown in Algorithm 16. Algorithm 17 applies a HS1 step to the graph G and to the associated h-graph $\mathcal{H}(G)$. Algorithm 18 adds a HS2 step such that v_3 is defined following Algorithm 19. Algorithm 19 shows an algorithm to compute an arbitrary v_3 element which preserves the tree-decomposability of the resulting graph, following Theorem 7.2.10.

Algorithm 16 Constructing a tree-decomposable Laman graph of a given order

```
Input: max-order, the desired order of the graph
       random, a function taking values 0 or 1 arbitrarily
Output: G, the tree-decomposable Laman graph with order max-order

function TreeDecomposableLaman()
G := Triangle Graph
HG := Hinges graph associated to G
while G.order < max-order do
  if random == 0 then
    add_H1_Vertex( G, HG )
  else
    add_H2_Vertex-TreeDec( G, HG )
  end if
end while
return G
endfunction
```

7.4 Conclusions

Hennberg steps and their relationship with tree-decomposability have been the object of study of this chapter. In Section 7.1 we have presented the definition of Henneberg steps, Henneberg sequences and Henneberg families, as well as a characterization of tree-decomposable Laman graphs based on an operation called merging. We have also established the inclusion relations of the Henneberg families with respect to the set of tree-decomposable Laman graphs.

In Section 7.2 we have studied the features of Henneberg steps, and established the conditions under which each kind of Henneberg step preserves the tree-decomposability of the graph in which it is applied. Two different results have been given for the case of

Algorithm 17 Add a HS1 to a tree-decomposable Laman graph

```
Input: G = (V, E), the tree-decomposable Laman subgraph,
       HG = (HV, ED, ES), the h-graph associated to G
Output: The modified graph G

function Add_H1_Vertex()
v1 := Arbitrary vertex in V
v2 := Arbitrary vertex in V different to v1
v := New vertex
V = V ∪ { v }
E = E ∪ { ( v, v1 ), ( v, v2 ) }
G = add_HS1( G, HG, v, v1, v2 )
return G
endfunction
```

Algorithm 18 Add a HS2 to a tree-decomposable Laman graph in a tree-decomposable way

```
Input: G = (V, E), the tree-decomposable Laman subgraph,
       HG = (HV, ED, ES), the h-graph associated to G
Output: The modified graph G

function Add_H2_Vertex-TreeDec()
e := Arbitrary edge in E
v1 := Source of e
v2 := Destination of e
v3 := Compute_Tree-Dec_V3(G, v1, v2)
v := New vertex
V = V ∪ { v }
E = E ∪ { ( v, v1 ), ( v, v2 ), ( v, v3 ) } \ { (v1, v2 ) }
G = add_HS2( G, HG, v, v1, v2, v3 )
return G
endfunction
```

Algorithm 19 Compute a third element for the HS2

Input: G = (V, E), the tree-decomposable Laman subgraph,
HG = (HV, ED, ES), the h-graph associated to G
e = (v1, v2), the edge involved in the HS2
Output: v3, a vertex assuring tree-decomposability of the HS2

function Compute_Tree-Dec_V3()
L := Set of nodes in HV including v1, v2
D := Set of nodes in HV which depend indirectly on a node in L
HV' = HV \ D
M := Maximal complete subgraph of HV' including the nodes in L
v3 := Arbitrary vertex of a node in M different to v1, v2
return v3
endfunction

HS2. The second one provides the tools for the development of an efficient algorithm for checking the tree-decomposability of the resulting graph.

Section 7.3 outlines a correct algorithm which generates tree-decomposable Laman graphs of a given order using Henneberg steps. The algorithm represents tree-decomposable Laman graphs as h-graphs, described in Chapter 3, and is based in the results proved in Section 7.2.

CHAPTER 8

Conclusions and future work

The future depends on what you do today.

Mahatma Gandhi

We present in this chapter the conclusions of our work, and we outline some open problems and work to be carried out in the future.

8.1 Conclusions

The reachability problem naturally arises in a number of computational processes where models are captured via geometry. Among them, our interest focuses on dynamic geometry and its applications to the development of current generation CAD/CAM systems.

Clearly dynamic geometry-based CAD/CAM systems improve over traditional parametric CAD/CAM systems by providing the user with the ability of dynamically exploring on the screen alternative solutions to the design problem. Dynamic geometry is no longer just geometry, it belongs to the dynamic systems field. Consequently tools developed in dynamic systems theory can help in solving open problems in dynamic geometry. In our approach we have successfully applied concepts like state variables and state transitions in

solving issues concerning continuity and conservatism.

Moving a variant parameter along a continuous path in a dynamic geometry system does not guarantee that geometric elements also follow a continuous path. The main sources of these problems are ambiguities. One source of ambiguity is the fact that, in general, geometric operations have more than one solution, for example, intersecting a line and a circle. Another ambiguity appears when a problem with a well-defined solution whenever geometric elements are in general position, say computing the point where two straight lines intersect, reaches a degenerate configuration, for example, the straight lines became parallel. To solve these problems, our approach fixes as a general requirement continuity in both the variant parameter and the geometric construction.

The solution to the dynamic geometry problem which fulfills continuity requirements does not need to be unique. Therefore an strategy to select the intended solution at each value of the variant parameter where more than one solution exist must be established. This selection is what actually defines the dynamic behavior of the geometric model. Clearly the especific strategy must be selected according to the problem goals. As an example of selection strategy, we have applied to select the behavior which minimizes the arc length of the variant parameter function. Other selection strategies may be however considered. Examples of strategies we could think of are to select the behavior such that: i) minimizes the path traced by a geometric articulation or ii) a given articulation is forced to follow a straight path or iii) two straight edges describe a minimal angle or iv) geometric elements are always placed within a predefined area in the operational space.

In this work we have proposed a technique to solve the reachability problem in dynamic geometry environments. In particular, geometric constructions based on constraints with one variant parameter are considered. This technique finds, if one exists, a continuous path from a given starting geometric configuration to a given ending one. The technique has been implemented on top of a dynamic geometry system based on constructive geometric constraint solving. Experimental results show that the approach is both effective and efficient from a practical point of view.

A procedure to solve the tracing problem has been also presented. The technique allows the user to define the especific movement the construction will follow, based on the bahavior he is expecting to see. The technique has been implemented on top of the dynamic geometry system based on constructive geometric constraint solving and deals efficiently with continuity and determinism at the same time.

The technique presented here to solve the reachability and tracing problems assumes the existence of a construction plan and is based on the analysis of the problem domain and the continuous transitions among domain intervals. The technique is divided into three steps. The first step of the methodology computes the domain of the variant parameter, which captures the set of feasible, unfeasible, and critical values. The computation of the

domain is performed using the method introduced by van der Meiden *et al.* in [103]. We have described a specific implementation of the van der Meiden *et al.* approach, developed on top of a ruler-and-compass geometric constraint solver, and we have presented for the first time a complete proof for the correctness of this method. It is limited to 2D problems but considers problems including points, straight lines and circles as geometric elements and point-point distance, perpendicular-straight line distance, line-line angle, point-on-circle and line-circle tangency. This set up enlarges the class of problems considered by van der Meiden, [105] and by Gao and Sitharam, [29].

In the second step the approach computes the transitions graph of the geometric constraint problem under study. The points computed by the van der Meiden method are also used to search for continuous transitions among the endpoints of the domain intervals. Continuous transitions allow the existence of dynamic evaluations which result in a continuous behavior. The transitions graph captures the set of continuous transitions among the domain intervals.

In the third step of the approach, the transitions graph is used to solve the problem at hand. In the case of the reachability problem, the A* algorithm is applied to search for a minimum path through the transitions graph. A continuous dynamic evaluation which solves the reachability problem is finally output. In the case of the tracing problem, the user is asked to define the specific behavior he is expecting to see, that is, the intended solution.

We have applied our prototype to a large benchmark of constraint problems. Running time allowed always real time interaction. However, a naive analysis of the worst case running time yields that time complexity of the approach is exponential on the number of signs in the index. Just notice that, in order to find the domain of the problem, the algorithm needs to check all the possible combinations of signs in the corresponding construction subplan.

We have also introduced h-graphs, a novel representation for tree-decomposable Laman graphs which captures in the same graph information of the graph and its tree-decomposition. We have also presented the basic features of h-graphs, which have been used for the implementation of our approach to solve the reachability and tracing problems. In particular, h-graphs allow to compute efficiently the kind of dependence of each tree-decomposition step with respect to the variant parameter.

Finally, we have developed a full study on Henneberg constructions and tree-decomposability, establishing the conditions under which tree-decomposition is guaranteed after the application of a Henneberg step. The results have been used to develop an algorithm which generates tree-decomposable Laman graphs of a given order by means of Henneberg steps.

8.2 Future work

We have developed our approach assuming that the variant parameter is arbitrarily chosen by the user. Concerning this point, the question of whether there is a strategy to find the *best* parameter naturally arises. This is an open and challenging problem which, as far as we know, nobody has studied yet.

If circles are excluded, extending our implementation to 3D problems would require replacing the underlying geometric constraint solver with another similar to the one used by van der Meiden *et al.* in [105]. A more general extension would require considering, for example, planes as basic geometric objects and the associated geometric constraints. A more difficult question arises from the fact that constructive geometric constraint solving in 3D entails hard open problems related to basic construction steps. These limitations and the technology available prevent the development of general constructive 3D solvers. See Hoffmann *et al.*, [44].

The main drawback of the technique presented in this work to solve the reachability and tracing problems comes from the fact that the van der Meiden method requires to tranform the problem at hand into a new geometric constraint problem for each indirectly dependent tree-decomposition step. This is a disadvantage for two different reasons. Firstly, to analyze and construct a problem is a very expensive process which increases the running time of the method dramatically. Secondly, the more problems to analyze the method has, the more probability there is to find a non-decomposable problem.

To deal with the need of transforming the problem, a simple procedure which may decrease the order of the modified problem and improve the efficiency of the method when analyzing and constructing the transformed problems could be considered. The van der Meiden method to compute the problem's domain analyzes all tree-decomposition steps which depend on the variant parameter and applies two different procedures depending on the kind of dependence arising.

For indirectly dependent tree-decomposition steps, the process includes the replacement of the original variant parameter by a new variant parameter which assures the direct dependence of the tree-decomosition step at hand, the construction of the new problem for each of its critical values, and the measure of the original variant parameter at each possible construction. In this process, the placements of the elements upon which neither the original variant parameter nor the new one are defined is irrelevant. Consider the original problem Π, represented by the graph $G = (V, E)$, and let the associated h-graph be $\mathcal{H}(G) = (\mathcal{V}, E_D, E_S)$. Assume that the original variant parameter is $\lambda = (u, v)$, which must be replaced by $\mu = (u', v')$. Using the h-graph and a variation of the procedure given in Section 3.5, Chapter 3, we can find the minimum complete subgraph of $\mathcal{H}(G)$ spanned by the geometric elements u, v, u', v', called M. Clearly, $|V(M)| \leq |V|$. Then, we

can apply the van der Meiden method to M. An analysis of the performance of this new approach would be necessary in order to determine the actual improvement with respect to the original method.

Concerning the non solvability of the transformed problem, Sitharam *et al.*, [97, 98], have presented an analysis based on Cayley configuration spaces of 1-dof tree-decomposable graphs. Following the notation in these works, the van der Meiden approach to the computation of the parameter ranges clearly entails to solve all the extreme graphs associated to the tree-decomposition steps which depend indirectly on the variant parameter. Graphs which assures the tree-decomposability of all these new graphs are actually those with low Cayley complexity. Sitharam *et al.* present a characterization of graphs with low Cayley complexity leading to an efficient algorithmic characterization. However, the highly demanding requirements of the characterization restrict the practical applicability of the theoretical results. A study of the relations arising between the van der Meiden method and low Cayley complexity would improve the applicability of the method reported here.

Bibliography

[1] B. Aldefeld. Variation of geometric based on a geometric-reasoning method. *Computer-Aided Design*, 20(3):117–126, April 1988.

[2] Hannah Bast. Efficient algorithms. In Susanne Albers, Helmut Alt, and Stefan Näher, editors, *Efficient Algorithms*, chapter Car or Public Transport–Two Worlds, pages 355–367. Springer-Verlag, Berlin, Heidelberg, 2009.

[3] Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast routing in very large public transportation networks using transfer patterns. In *Proceedings of the 18th annual European conference on Algorithms: Part I*, ESA'10, pages 290–301, Berlin, Heidelberg, 2010. Springer-Verlag.

[4] Y. Baulac, F. Bellemain, and J.-M. Laborde. *Cabri - the Interactive Geometry Notebook*. Brooks/Cole Publishing Company, 1992.

[5] Bernhard Bettig and Christoph M. Hoffmann. Geometric constraint solving in parametric CAD. *Journal of Computing and Information Science in Engineering*, 11(2), 2011.

[6] John Adrian Bondy and U.S.R. Murty. *Graph Theory with Applications*. The Macmillan Press Ltd., 1982.

[7] Ciprian Borcea and Ileana Streinu. The number of embeddings of minimally rigid graphs. *Discrete & Computational Geometry*, 31:287–303, 2004.

[8] W. Bouma, I. Fudos, C. Hoffmann, J. Cai, and R. Paige. A geometric constraint solver. Technical Report CSD-TR-93-054, Department of Computer Scineces, Purdue University, 1993.

[9] W. Bouma, I. Fudos, C. Hoffmann, J. Cai, and R. Paige. Geometric constraint solver. *Computer Aided Design*, 27(6):487–501, June 1995.

[10] Olivier Bournez and Igor Potapov, editors. *Reachability Problems. 3rd International Workshop, RP 2009, Palaiseau, France, September 23-25, 2009. Proceedings*, volume 5797 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009.

[11] B.D. Brüderlin. Symbolic computer geometry for computer aided geometric design. In *Advances in Design and Manufacturing Systems*, Tempe, AZ, Jan. 8-12 1990. Proceedings NSF Conference.

[12] B. Buchberger. *Multidimensional Systems Theory*, chapter Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory, pages 184–232. D. Reidel Publishing Theory, 1985.

[13] Nicos Christofides. *Graph Thoery: An Algorithmic Approach*. Academic Press Inc. LTD., 21/28 Oval Road. London NW1, 1975.

[14] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the Association for Computing Machinery*, 32(3):505–536, July 1985.

[15] B. Denner-Broser. *Automated Deduction in Geometry*, chapter On the Decidability of Tracing Problems in Dynamic Geometry, pages 111–129. Springer, 2006.

[16] B. Denner-Broser. *Tracing-Problems in Dynamic Geometry*. PhD thesis, Institut für Informatik, Freie Universität Berlin, Takustrasse 9, 14195 Berlin, 2008.

[17] Britta Denner-Broser. An algorithm for the tracing problem using interval analysis. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 1832–1837, New York, NY, USA, 2008. ACM.

[18] Britta Denner-Broser. About tracing problems in dynamic geometry. *Discrete & Computational Geometry*, 49:221–246, 2013.

[19] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[20] C. Durand. *Symbolic and Numerical Techniques for Constraint Solving*. PhD thesis, Computer Science, Purdue University, December 1998.

[21] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics Principles and Practice*. Addison-Wesley Pu. Co., 2nd edition, 1996.

[22] András Frank and László Szegö. An extension of a theorem of Henneberg and Laman. Technical Report TR-2001-05, Egérvary Research Group on Combinatorial Optimization, www.cs.elte.hu/egres, February 2001.

[23] M. Freixas, R. Joan-Arinyo, A. Soto-Riera, and S. Vila-Marta. SolBCN a constraint-based two dimensional geometric editor, february 2008. http://floss.lsi.upc.edu/wiki/solBCN.

[24] Marc Freixas, Robert Joan Arinyo, and Antoni Soto-Riera. A constraint-based dynamic geometry system. In *Proceedings of the 2008 ACM symposium on Solid and physical modeling*, SPM '08, pages 37–46, New York, NY, USA, 2008. ACM.

[25] Marc Freixas, Robert Joan-Arinyo, and Antoni Soto-Riera. A constraint-based dynamic geometry system. *Computer Aided Design*, 42(2):151–161, February 2010.

[26] I. Fudos. *Constraint Solving for Computer Aided Design*. PhD thesis, Purdue University, Department of Computer Sciences, 1995.

[27] I. Fudos and C.M. Hoffmann. Correctness proof of a geometric constraint solver. *International Journal of Computational Geometry and Applications*, 6(4):405–420, 1996.

[28] I. Fudos and C.M. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics*, 16(2):179–216, April 1997.

[29] Heping Gao and Meera Sitharam. Characterizing 1-dof henneberg-I graphs with efficient configuration spaces. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1122–1126, 2009.

[30] GeoGebra. http://www.geogebra.org/en/wiki/index.php, 2010.

[31] GeoGebra. http://www.geogebra.org/cms, July 2007.

[32] Reginald G. Golledge, Roberta L. Klatzky, Jack M. Loomis, Jon Speigle, and Jerome Tietz. A geographical information system for a GPS based personal guidance system. *International Journal of Geographical Information Science*, 12(7):727–749, 1998.

[33] Jonathan L. Gross and Jay Yellen. *Handbook of Graph Theory*. CRC Press LLC, 2000 N.W. Corporate Blvd. Boca Raton, Florida 33431, 2004.

[34] R. Haas, D. Orden, G. Rote, F. Santos, B. Servatius, H. Servatius, D. souvaine souvaine souvaine souvaine, I. Streinu, and W. Whiteley. Planar minimally rigid graphs and pseudo-triangulations. *Computational Geometry*, 31(1-2):31–61, 2005.

[35] L. Henneberg. *Die graphische Statik der starren Systeme*. B. G. Teubner, 1911.

[36] A. Heydon and G. Nelson. The Juno-2 constraint-based drawing editor. Research Report 131a, Digital Systems Research Center, December 1994.

[37] Marta Hidalgo and Robert Joan-Arinyo. Computing parameter ranges in constructive geometric constraint solving: Implementation and correctness proof. *Computer Aided Design*, 44(7):709–720, July 2012.

[38] Marta R. Hidalgo and Robert Joan-Arinyo. On continuity in geometric constraint-based dynamic geometry. In Lidia Ortega and Alejandro León, editors, *Aplicación de Herramientas CAD a Realidad Virtual: Representaciones Jerárquicas y Luces Virtuales*, chapter On Continuity in Geometric Constraint-Based Dynamic Geometry, pages 13–16. CopiCentro Editorial, 2010.

[39] Marta R. Hidalgo and Robert Joan-Arinyo. The reachability problem in constructive geometric constraint solving based dynamic geometry. Technical report, Department LSI, Universitat Politècnica de Catalunya, 2011.

[40] Marta R. Hidalgo and Robert Joan-Arinyo. The reachability problem in constructive geometric constraint solving based dynamic geometry. *Journal of Automated Reasoning*, pages 1–24, 2013.

[41] Marta R. Hidalgo, Robert Joan-Arinyo, and Antoni Soto-Riera. Geometric constraint problems and solution instances. Technical report, Department LSI, Universitat Politècnica de Catalunya, 2010.

[42] Marta R. Hidalgo, Robert Joan-Arinyo, and Antoni Soto-Riera. Computing parameter ranges in constructive geometric constraint solving. A correctness proof. Technical report, Department LSI, Universitat Politècnica de Catalunya, 2011.

[43] C.M. Hoffmann and R. Joan-Arinyo. Symbolic constraints in constructive geometric constraint solving. *Journal of Symbolic Computation*, 23:287–300, 1997.

[44] C.M. Hoffmann and R. Joan-Arinyo. Distributed maintenance of multiple product views. *Computer-Aided Design*, 32(7):421–431, June 2000.

[45] C.M. Hoffmann and R. Joan-Arinyo. A brief on constraint solving. *Computer-Aided Design and Applications*, 2(5):655–663, 2005.

[46] C.M. Hoffmann and K.J. Kim. Towards valid parametric CAD models. *Computer-Aided Design*, 33(1):376–408, 2001.

[47] C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Decompostion Plans for Geometric Constraint Problems, Part II: New Algorithms. *Journal of Symbolic Computation*, 31:409–427, 2001.

[48] C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Decompostion Plans for Geometric Constraint Systems, Part I: Performance Measurements for CAD. *Journal of Symbolic Computation*, 31:367–408, 2001.

[49] C.M. Hoffmann and P.J. Vermeer. Geometric constraint solving in R^2 and R^3. In D.-Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry*, pages 266–298. World Scientific Publishing, 1995.

[50] C.M. Hoffmann and P.J. Vermeer. A spatial constraint problem. In J.P. Merlet and B. Ravani, editors, *Computational Kinematics'95*, pages 83–92. Kluwer Academic Publ., 1995.

[51] R. Hölzl. How does 'dragging' affect the learning of geometry. *International Journal of Computers for Mathematical Learning*, Volume 1(1):169–187, Januar 2001.

[52] C.-Y. Hsu. *Graph-Based Approach for Solving Geometric Constraint Problems*. PhD thesis, Department of Computer Science. The University of Utah, June 1996.

[53] C.-Y. Hsu and B.D. Brüderlin. A hybrid constraint solver using exact and iterative geometric constructions. In D. Roller and P. Brunet, editors, *CAD Systems Development: Tools and Methods*, pages 265–279, Berlin, 1997. Springer-Verlag.

[54] Yong K. Hwang and Narendra Ahuja. Gross motion planning: A survey. *ACM Computer Surveys (CSUR)*, 24(3):219–291, September 1992.

[55] B. Servatius H. Servatius J. Graver. *Combinatorial Rigidity*. J. E. Humphreys, R. C. Kirby, L. W. Small, 1993.

[56] N. Jackiw. *The Geometer's Sketchpad*. Key Curriculum Press, Berkeley, 1991–1995.

[57] N. Jackiw. Visualizing complex functions with the geometer's sketchpad. In T. Triandafillidis and Eds. Hatzikiriakou, editors, *Proceedings of the 6th International Conference on Technology in Mathematics Teaching*, pages 291–299, 2003.

[58] Bill Jackson and Tibor Jordán. Globally rigid circuits of the direction–length rigidity matroid. *Journal of Combinatorial Theory, Series B*, 100(1):1–22, January 2010.

[59] C. Jerman, G. Trombettoni, B. Neveu, and P. Mathis. Decomposition of geometric constraint systems: A survey. *International Journal of Computational Geometry and Applications*, 16:379–414, 2006.

[60] C. Jermann. *Résolution de constraintes géométriques par rigidification récursive et propagation d'intervalles*. PhD thesis, Université de Nice-Sophia Antipolis, 2002. (Written in French).

[61] R. Joan-Arinyo, M. V. Luzón, and E. Yeguas. Search space pruning to solve the root identification problem in geometric constraint solving. *Computer-Aided Design and Applications*, 6(1):15–25, 2009.

[62] R. Joan-Arinyo, M.V. Luzón, and A. Soto. Genetic algorithms for root multiselection in constructive geometric constraint solving. *Computers & Graphics*, 27(1):51–60, 2003.

[63] R. Joan-Arinyo and N. Mata. Applying constructive geometric constraint solvers to geometric problems with interval parameters. *Nonlinear Analysis*, 47(1):213–224, 2001.

[64] R. Joan-Arinyo and A. Soto. A correct rule-based geometric constraint solver. *Computer & Graphics*, 21(5):599–609, 1997.

[65] R. Joan-Arinyo and A. Soto-Riera. Combining constructive and equational geometric constraint solving techniques. *ACM Transactions on Graphics*, 18(1):35–55, January 1999.

[66] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana. On the domain of constructive geometric constraint solving techniques. In R. Duricovic and S. Czanner, editors, *Spring Conference on Computer Graphics*, pages 49–54, Budmerice, Slovakia, April 25-28 2001. IEEE Computer Society, Los Alamitos, CA.

[67] L. W. Johnson and R. D. Riess. *Numerical analysis*. Addison-Wesley, 1982. Second edition.

[68] Tibor Jordan and Zoltan Szabadka. Operations preserving the global rigidity of graphs and frameworks in the plane. *Computational Geometry*, 42(6 - 7):511–521, 2009.

[69] M.-W. Kang, M.K. Pha, and D. Hwang. Part I. A GIS-based simulation model for positioning and routing unmanned ground vehicles. Technical report, Center for Advanced Transportation and Infrastructure Engineering, Morgan State University, 2010.

[70] U. Kortenkamp. *Foundations of Dynamic Geometry*. PhD thesis, Swiss Federal Institute of Technology Zurich, 1999.

[71] G. Kramer. *Solving Geometric Constraints Systems*. MIT Press, 1992.

[72] G.A. Kramer. Using degrees of freedom analysis to solve geometric constraint systems. In J. Rossignac and J. Turner, editors, *Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 371–378, Austin, TX, June 5-7 1991. ACM Press.

[73] G.A. Kramer. A geometric constraint engine. *Artificial Intelligence*, 58(1-3):327–360, 1992.

[74] J.-M. Laborde and F. Bellemain. *Cabri-Geometry II*. Texas Instruments and Université Joseph Fourier, 1993–1998.

[75] G. Laman. On graphs and rigidity of plane skeletal structures. *Journal of Engineering Mathematics*, 4(4):331–340, October 1970.

[76] Jean-Paul Laumond. Motion planning for PLM: state of the art and perspectives. *International Journal of Product Lifecycle Management*, 1(2):129–142, 2006.

[77] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[78] R. Light and D. Gossard. Modification of geometric models through variational geometry. *Computer Aided Design*, 14:209–214, July 1982.

[79] V.C. Lin, D.C. Gossard, and R.A. Light. Variational geometry in computer-aided design. *ACM Compuer Graphics*, 15(3):171–177, August 1981.

[80] Ernst W. Mayr. An algorithm for the general petri net reachability problem. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, STOC '81, pages 238–246, New York, NY, USA, 1981. ACM.

[81] T Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[82] G. Nelson. Juno, a constraint-based graphics system. *SIGGRAPH*, pages 235–243, San Francisco, July 22–26 1985.

[83] Thorsten Orendt. *Resolution of Geometric Singularities by Complex Detours - Modeling, Complexity and Application*. PhD thesis, Zentrum Mathematik, Technische Universität München, 2010.

[84] J.C. Owen. Algebraic solution for geometry from dimensional constraints. In R. Rossignac and J. Turner, editors, *Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 397–407, Austin, TX, June 5-7 1991. ACM Press.

[85] James L. Peterson. *Petri Net Theory and the Modelling of Systems*. Prentice Hall Inc., Englewood Cliffs, N.J. 07632, 1981.

[86] Srinivas Raghothama and Vadim Shapiro. Necessary conditions for boundary representation variance. In *Proceedings of the 13th annual symposium on Computational geometry*, SCG '97, pages 77–86, New York, NY, USA, 1997. ACM.

[87] Srinivas Raghothama and Vadim Shapiro. Boundary representation deformation in parametric solid modeling. *ACM Transactions on Graphics*, 17(4):259–286, October 1998.

[88] Srinivas Raghothama and Vadim Shapiro. Consistent updates in dual representation systems. In *Proceedings of the fifth ACM symposium on Solid modeling and applications*, SMA '99, pages 65–75, New York, NY, USA, 1999. ACM.

[89] J. Richter-Geber and U. Kortenkamp. *The interactive geometry software Cinderella*. Springer-Verlag, 1999.

[90] Jürgen Richter-Gebert and Ulrich H. Kortenkamp. Complexity issues in dynamic geometry. In *Proceedings of the Smale Fest 2000, Hong Kong*, pages 193–198, 2000.

[91] Stuart Russell and Peter Norvig. *Artificial Intelligence: a modern approach*. Alan Apt, 1995.

[92] Peter Sanders and Dominik Schultes. Highway hierarchies hasten exact shortest path queries. In *Proceedings of the 13th annual European conference on Algorithms*, ESA'05, pages 568–579, Berlin, Heidelberg, 2005. Springer-Verlag.

[93] Peter Sanders and Dominik Schultes. Engineering fast route planning algorithms. In *Proceedings of the 6th international conference on Experimental algorithms*, WEA'07, pages 23–36, Berlin, Heidelberg, 2007. Springer-Verlag.

[94] Brigitte Servatius and Herman Servatius. Rigidity, global rigidity, and graph decomposition. *European Journal of Combinatorics*, 31(4):1121–1135, 2010. Rigidity and Related Topics in Geometry.

[95] Stephan Seufert, Avishek Anand, Srikanta J. Bedathur, and Gerhard Weikum. FERRARI: Flexible and Efficient Reachability Range Assignment for Graph Indexing. In *ICDE'13: Proceedings of the 29th IEEE International Conference on Data Engineering*. IEEE, 2013.

[96] Vadim Shapiro and Donald L. Vossler. What is a parametric family of solids? In *Proceedings of the third ACM symposium on Solid modeling and applications*, SMA '95, pages 43–54, New York, NY, USA, 1995. ACM.

[97] Meera Sitharam, Menghan Wang, and Heping Gao. Cayley configuration spaces of 1-dof tree-decomposable linkages, part I: Structure and extreme points. *CoRR*, abs/1112.6008, 2011.

[98] Meera Sitharam, Menghan Wang, and Heping Gao. Cayley configuration spaces of 1-dof tree-decomposable linkages, part II: Combinatorial characterization of complexity. *CoRR*, abs/1112.6009, 2011.

[99] A. Soto. *Satisfacció de restriccions geomètriques en 2D*. PhD thesis, Universitat Politècnica de Catalunya, Dept. Llenguatges i Sistemes Informàtics, 1998. (Written in Catalan).

[100] Tiong-Seng Tay and Walter Whiteley. Generating isostatic frameworks. *Structural Topology*, (11), 1985.

[101] S.E.B. Thierry. *Décomposition et paramétrisation de systèmes de contraites géométriques sous-constraints*. PhD thesis, Université de Strasbourg, 2010.

[102] P. Todd. A k-tree generalization that characterizes consistency of dimensioned engineering drawings. *SIAM Journal on Discrete Mathematics*, 2(2):255–261, 1989.

[103] H.A. van der Meiden. *Semantics of Families of Objects*. PhD thesis, Delft University of Technology, The Nederlands, 2008.

[104] H.A. van der Meiden and W.F. Bronsvoort. An efficient method to determine the intended solution for a system of geometric constraints. *International Journal of Computational Geometry*, 15(3):79–98, 2005.

[105] H.A. van der Meiden and W.F. Bronsvoort. A constructive approach to calculate parameter ranges for systems of geometric constraints. *Computer-Aided Design*, 38:275–283, 2006.

[106] S. Vila. *Contribution to Geometric Constraint Solving in Cooperative Engineering*. PhD thesis, Departament Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, 2003.

[107] W. Whiteley. *Handbook of discrete and computational geometry*, chapter Rigidity and scene analysis, pages 893–916. CRC Press, Inc., Boca Raton, FL, USA, 1997.

[108] Harald Winroth. Dynamic projective geometry. Master's thesis, Stockholms Universitet, KTH, Numerical Analysis and Computer Science, 1999.

[109] Jing Yang, Patrick Dymond, and Michael Jenkin. Exploiting hierarchical probabilistic motion planning for robot reachable workspace estimation. In JuanAndrade Cetto, Joaquim Filipe, and Jean-Louis Ferrier, editors, *Informatics in Control Automation and Robotics*, volume 85 of *Lecture Notes in Electrical Engineering*, pages 229–241. Springer Berlin Heidelberg, 2011.

[110] Xinming Ye, Jiantao Zhou, and Xiaoyu Song. On reachability graphs of Petri nets. *Computers & Electrical Engineering*, 29(2):263–272, 2003.

[111] Hilmi Yildirim, Vineet Chaoji, and Mohammed J. Zaki. GRAIL: A scalable index for reachability queries in very large graphs. *VLDB Journal*, pages 509–534, 2012.

[112] Zhiyuan Ying and S.Sitharama Iyengar. Robot reachability problem: A nonlinear optimization approach. *Journal of Intelligent and Robotic Systems*, 12:87–100, 1995.